Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 191.

Fourth Edition (April 2005) (Softcopy Only)

This edition replaces or makes obsolete the previous edition, SC27-1292-02. This edition is available in softcopy format only. The technical changes for this version are summarized under "Summary of Changes" on page xv.

© Copyright International Business Machines Corporation 1997, 2005. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
# Contents

**Figures** .......................................................... vii  
**Tables** .......................................................... ix  

**About This Book** ............................................ xi  
- Prerequisite Knowledge ........................................ xi  
- How to Read Syntax Diagrams .............................. xi  
  - Example Syntax Diagram ................................. xiii  
- How to Send Your Comments .............................. xiv  

**Summary of Changes** ...................................... xv  
- Changes to the Current Edition of This Book for IMS Version 8 ........................................ xv  
- Changes to This Book for IMS Version 8 .............. xv  
- Library Changes for IMS Version 8. .................... xv  
  - New, Revised, and Eliminated Titles .................. xvi  
  - Organizational Changes ................................ xvi  
  - Deleted Information ..................................... xvi  
  - Accessibility Enhancements ............................ xvii  

**Chapter 1. Introduction to the Common Queue Server** ............ 1  
- Terms Used in This Book .................................. 1  
- Common Queue Server Overview .......................... 4  
  - CQS Benefits ............................................. 5  
  - CQS Components ........................................ 5  
  - CQS Functions ........................................... 6  
  - Structures Managed by CQS ............................ 6  
  - CQS Structure Functions ............................... 7  
  - CQS Recovery Functions ............................... 8  
  - CQS Client Requests .................................. 9  
- Planning for CQS Hardware and Software Requirements .......... 9  

**Chapter 2. CQS Definition and Tailoring** ......................... 11  
- CQS As Part of a Sysplex ................................ 11  
- CQS and Defining MVS Policies ........................... 11  
- CQS's Support of Multiple Clients ....................... 14  
- Determining Structure Size for CQS Connections .......... 15  
- Preparing to Start CQS .................................... 15  
- Updating MVS Program Properties Table for CQS .......... 16  
- CQS Execution Parameters ................................. 17  
- CQS Initialization Parameters PROCLIB Member (CQSIPxxx) .............. 19  
- CQS Local Structure Definition PROCLIB Member (CQSSLxxx) .......... 20  
- CQS Global Structure Definition PROCLIB Member (CQSSGxxx) .......... 22  
- CQS Execution Data Sets .................................. 27  
  - CQS System Checkpoint Data Set ...................... 27  
  - CQS Structure Recovery Data Sets .................... 28  

**Chapter 3. CQS Administration** ................................ 31  
- Starting CQS .................................................. 31  
- Recording Information Necessary for Starting CQS .......... 32  
- Restarting CQS Structures .................................. 32  
  - CQS Structure Allocation ............................... 32  
  - CQS Structure Warm Start .............................. 32  
  - CQS Structure Cold Start .............................. 33  

© Copyright IBM Corp. 1997, 2005
Chapter 5. Writing a CQS Client ........................................ 73
Introducing CQS Client Requests ...................................... 73
Sequence of CQS Requests Issued by a Client for Queue Structure .............................. 74
Coding CQS Requests .................................................. 74
Authorization for CQS .................................................. 74
Environmental Requirements for CQS .................................. 75
Using Registers with CQS Requests .................................... 76
Coding Parameters for CQS Requests .................................. 76
Using an ECB with CQS Requests ...................................... 78
Using Lists in the CQS Requests ....................................... 79
Return Codes and Reason Codes for CQS Requests ..................... 79
Assembling a Program with CQS Requests .............................. 81
CQS Clients and Handling Special Events ............................... 81
CQS Cold Start ......................................................... 81
Registering Interest in Queues with CQSINFRM ......................... 82
Working with Objects on the Cold Queue using CQS Requests ............ 82
Initiating Checkpoints using CQS Requests ............................. 82
Shutting Down CQS .................................................. 82
Tuning to Improve CQS Performance ................................... 82

Chapter 6. CQS Client Requests ........................................ 85
Using CQS Client Requests ............................................. 85
CQSBRWSE Request .................................................. 86
CQSHCKPT Request .................................................. 93
CQSCONN Request .................................................. 96
CQSDEL Request ..................................................... 102
CQSDEREG Request .................................................. 106
CQSDISC Request ................................................... 107
CQSINFRM Request .................................................. 112
CQSMOVE Request ................................................... 116
CQSPUT Request ..................................................... 120
CQSQUERY Request .................................................. 127
CQSREAD Request ................................................... 136
CQSRRCVR Request .................................................. 141
CQSREG Request ..................................................... 146
CQSSSYNC Request .................................................. 148
CQSSHUT Request .................................................... 155
CQSUNLCK Request ................................................... 156
CQSUPD Request ..................................................... 161
Example of Using a CQS Request: CQSREAD ............................ 165

Chapter 7. CQS Client Exit Routines .................................... 171
Client CQS Event Exit Routine ......................................... 171
Contents of Registers on Entry .......................................... 171
Contents of Registers on Exit ........................................... 172
**Figures**

1. Client Systems, CQS, and the Coupling Facility ......................................... 5  
2. Defining IMS Resources in the CFRM Policy ................................................... 13  
3. Defining IMS Resources in the LOGR Policy .................................................. 14  
4. Defining IMS Resources in the SFM Policy ..................................................... 14  
5. Entry to Be Added to the MVS Program Properties Table .................................. 16  
6. Specifying IMS and CQS Parameters ................................................................. 18  
7. Sample CQSIPxxx PROCLIB Member ................................................................. 19  
8. Sample CQSSLxxx PROCLIB Member ................................................................. 21  
9. Sample CQSSGxxx PROCLIB Member ................................................................. 23  
10. System Checkpoint Data Set Example ............................................................. 28  
11. Structure Recovery Data Set Example ............................................................. 29  
12. RACF Commands for Authorizing CQS Registration ......................................... 37  
13. RACF Commands to Authorize Connection to CQS Structures ............................. 38  
14. Display for Structure Full Threshold - Example 1 .......................................... 42  
15. Display for Structure Full Threshold - Example 2 .......................................... 42  
16. Display for Structure Full Threshold - Example 3 .......................................... 43  
17. Passing an Address for Register .................................................................... 77  
18. Passing a Value for Register .............................................................................. 77  
19. Passing an Address for Symbol .......................................................................... 77  
20. Passing a Value for Symbol ................................................................................. 77  
21. Passing a Value for Symbol Value ....................................................................... 78  
22. Passing an Equate for Symbol Value ................................................................... 78  
23. Coding CQSREAD with the OPTWORD1 parameter ......................................... 78  
24. STEPLIB DD Statement to Concatenate IMS.SDFSRESL ...................................... 81  
25. Sample for CQSREAD Request ......................................................................... 166  
26. JCL to Print CQS Log Records ........................................................................... 187  
27. DD Card to Limit Log Records that are Printed ............................................... 188  
28. DD Card to Add Local Date and Time ................................................................. 188  
29. JCL to Copy CQS Records from a Specific Time Period ..................................... 189
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>How to Read Syntax Diagrams</td>
</tr>
<tr>
<td>2.</td>
<td>Private Queue Types Managed by CQS</td>
</tr>
<tr>
<td>3.</td>
<td>CQS Init-Term User-Supplied Exit Routine Parameter List: CQS Initialization</td>
</tr>
<tr>
<td>4.</td>
<td>CQS Init-Term User-Supplied Exit Routine Parameter List: CQS Termination</td>
</tr>
<tr>
<td>5.</td>
<td>CQS Client Connection User-Supplied Exit Routine Parameter List: Client Connection</td>
</tr>
<tr>
<td>6.</td>
<td>CQS Client Connection User-Supplied Exit Routine Parameter List: Client Disconnect</td>
</tr>
<tr>
<td>7.</td>
<td>CQS Queue Overflow User-Supplied Exit Routine Parameter List</td>
</tr>
<tr>
<td>8.</td>
<td>CQS Structure Statistics User-Supplied Exit Routine Parameter List</td>
</tr>
<tr>
<td>9.</td>
<td>CQS Structure Process Statistics Record</td>
</tr>
<tr>
<td>10.</td>
<td>CQS Request Statistics Record</td>
</tr>
<tr>
<td>11.</td>
<td>Data Object Statistics Record</td>
</tr>
<tr>
<td>12.</td>
<td>Queue Name Statistics Record</td>
</tr>
<tr>
<td>13.</td>
<td>MVS Request Statistics Record</td>
</tr>
<tr>
<td>14.</td>
<td>Structure Rebuild Statistics Record</td>
</tr>
<tr>
<td>15.</td>
<td>Structure Checkpoint Statistics Record</td>
</tr>
<tr>
<td>16.</td>
<td>Structure Checkpoint Statistics Entry</td>
</tr>
<tr>
<td>17.</td>
<td>CQS Structure Event User-Supplied Exit Routine Parameter List: Connect</td>
</tr>
<tr>
<td>18.</td>
<td>CQS Structure Event User-Supplied Exit Routine Parameter List: Checkpoint</td>
</tr>
<tr>
<td>19.</td>
<td>CQS Structure Event User-Supplied Exit Routine Parameter List: Rebuild</td>
</tr>
<tr>
<td>20.</td>
<td>CQS Structure Event User-Supplied Exit Routine Parameter List: Overflow</td>
</tr>
<tr>
<td>21.</td>
<td>CQS Structure Event User-Supplied Exit Routine Parameter List: Status Change</td>
</tr>
<tr>
<td>22.</td>
<td>CQS Statistics Header Data</td>
</tr>
<tr>
<td>23.</td>
<td>Sequence for CQS Requests</td>
</tr>
<tr>
<td>24.</td>
<td>Environment for CQS Requests (Excluding CQSREG and CQSDEREG) Using the Authorized Interface</td>
</tr>
<tr>
<td>25.</td>
<td>Environment for CQS Requests (Excluding CQSREG and CQSDEREG) Using the Non-Authorized Interface</td>
</tr>
<tr>
<td>26.</td>
<td>Environment for CQSREG and CQSDEREG Requests Using the Authorized Interface</td>
</tr>
<tr>
<td>27.</td>
<td>Environment for CQSREG and CQSDEREG Requests Using the Non-Authorized Interface</td>
</tr>
<tr>
<td>28.</td>
<td>Return and Reason Codes for Errors Detected by the CQS Interface</td>
</tr>
<tr>
<td>29.</td>
<td>CQSBRWSE Return and Reason Codes</td>
</tr>
<tr>
<td>30.</td>
<td>CQSCHKPT Return and Reason Codes</td>
</tr>
<tr>
<td>31.</td>
<td>CQSCONN Return and Reason Codes</td>
</tr>
<tr>
<td>32.</td>
<td>CQSDDEL Return and Reason Codes</td>
</tr>
<tr>
<td>33.</td>
<td>CQSDEREG Return and Reason Codes</td>
</tr>
<tr>
<td>34.</td>
<td>CQSDISC Return and Reason Codes</td>
</tr>
<tr>
<td>35.</td>
<td>CQSNFRM Return and Reason Codes</td>
</tr>
<tr>
<td>36.</td>
<td>CQSMOVE Return and Reason Codes</td>
</tr>
<tr>
<td>37.</td>
<td>Actions Taken for Data Objects as a Result of Failures or Structure Activity</td>
</tr>
<tr>
<td>38.</td>
<td>CQSPUT Return and Reason Codes</td>
</tr>
<tr>
<td>39.</td>
<td>CQSQUERY Return and Reason Codes</td>
</tr>
<tr>
<td>40.</td>
<td>CQREAD Return and Reason Codes</td>
</tr>
<tr>
<td>41.</td>
<td>CQRECVR Return and Reason Codes</td>
</tr>
<tr>
<td>42.</td>
<td>CQSREG Return and Reason Codes</td>
</tr>
<tr>
<td>43.</td>
<td>UOW Status from the Client</td>
</tr>
<tr>
<td>44.</td>
<td>UOW Status from CQS</td>
</tr>
<tr>
<td>45.</td>
<td>CQSSSYNC Return and Reason Codes</td>
</tr>
<tr>
<td>46.</td>
<td>CQSSHUT Return and Reason Codes</td>
</tr>
<tr>
<td>47.</td>
<td>CQSUNLCK Return and Reason Codes</td>
</tr>
<tr>
<td>48.</td>
<td>CQSUPD Return and Reason Codes</td>
</tr>
<tr>
<td>49.</td>
<td>Client CQS Event Exit Routine Parameter List: CQS Restart Entry</td>
</tr>
<tr>
<td>50.</td>
<td>Client CQS Event Exit Routine Parameter List: CQS Abnormal Termination</td>
</tr>
<tr>
<td>51.</td>
<td>Client Structure Event Exit Routine Parameter List: Deferred Resync Complete</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1997, 2005
52. Client Structure Event Routine Exit Parameter List: CQS Initiated Resync ..................... 176
53. CQS Resync UOW Entry Parameters ......................................................... 176
54. Client Structure Event Exit Routine Parameter List: Checkpoint ......................... 177
56. Client Structure Event Exit Routine Parameter List: Structure Rebuild Lost UOWs .... 179
57. CQS Rebuild Lost UOW Entry Parameters .................................................. 179
58. Client Structure Event Exit Routine Parameter List: Structure Overflow ............... 180
60. Client Structure Inform Exit Routine Parameter List ......................................... 182
61. CQS Log Records ..................................................................................... 185
About This Book

This book is designed to help programmers, operators, and system support personnel perform these tasks:

- Plan for and design the installation of Common Queue Server (CQS).
- Install and operate CQS.
- Diagnose and recover from CQS system problems.
- Write a CQS client.

This information is available in PDF and BookManager® formats, and also as part of the DB2® Information Management Software Information Center for z/OS® Solutions. To view the information within the DB2 Information Management Software Information Center for z/OS Solutions, go to publib.boulder.ibm.com/infocenter/dzichelp. To get the most current versions of the PDF and BookManager formats, go to the IMS™ Library page at www.ibm.com/software/data/ims/library.html.

Prerequisite Knowledge

Before using this book, you should understand:

- Basic IMS concepts
- The IMS environment
- Coupling facility configuration concepts
- Sysplex configuration concepts

For a list of references to related publications, refer to "Bibliography" on page 195.

Related Reading: For definitions of terminology specific to CQS and used in this manual, see Chapter 1, "Introduction to the Common Queue Server," on page 1. Other terms are defined in the IMS Version 8: Master Index and Glossary.

How to Read Syntax Diagrams

This book contains syntax diagrams.

Each syntax diagram begins with a double right arrow and ends with a right and left arrow pair. Lines that begin with a single right arrow are continuation lines. You read a syntax diagram from left to right and from top to bottom, following the direction of the arrows.

Table 1 describes the conventions that are used in syntax diagrams in this information:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>A – B – C</td>
<td>You must specify values A, B, and C. Required values are shown on the main path of a syntax diagram.</td>
</tr>
<tr>
<td>Convention</td>
<td>Meaning</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td><img src="image1.png" alt="Diagram" /></td>
<td>You must specify value A, B, or C.</td>
</tr>
<tr>
<td><img src="image2.png" alt="Diagram" /></td>
<td>You have the option to specify value A. Optional values are shown below the main path of a syntax diagram.</td>
</tr>
<tr>
<td><img src="image3.png" alt="Diagram" /></td>
<td>You have the option to specify A, B, C, or none of these values.</td>
</tr>
<tr>
<td><img src="image4.png" alt="Diagram" /></td>
<td>You have the option to specify A, B, C, or none of these values. If you don’t specify a value, A is the default.</td>
</tr>
<tr>
<td><img src="image5.png" alt="Diagram" /></td>
<td>You have the option to specify one, more than one, or none of the values A, B, or C. Any required separator for multiple or repeated values (in this example, the comma) is shown on the arrow.</td>
</tr>
<tr>
<td><img src="image6.png" alt="Diagram" /></td>
<td>You have the option to specify value A multiple times. The separator in this example is optional.</td>
</tr>
<tr>
<td><img src="image7.png" alt="Diagram" /></td>
<td>Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.</td>
</tr>
</tbody>
</table>

**Name:**

<table>
<thead>
<tr>
<th>Name:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image8.png" alt="Diagram" /></td>
<td></td>
</tr>
</tbody>
</table>

**Punctuation marks and numbers**

Enter punctuation marks (slashes, commas, periods, parentheses, quotation marks, equal signs) and numbers exactly as shown.

<table>
<thead>
<tr>
<th>Uppercase values</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image9.png" alt="Diagram" /></td>
<td>Keywords, their allowable synonyms, and reserved parameters appear in uppercase letters for OS/390®. Enter these values exactly as shown.</td>
</tr>
</tbody>
</table>
Table 1. How to Read Syntax Diagrams (continued)

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase values</td>
<td>Keywords, their allowable synonyms, and reserved parameters appear in lowercase letters for UNIX®. Enter these values exactly as shown.</td>
</tr>
<tr>
<td>Lowercase values in italic (for example, name)</td>
<td>Supply your own text or value in place of the name variable.</td>
</tr>
<tr>
<td>\textbackslash b</td>
<td>A \textbackslash b symbol indicates one blank position.</td>
</tr>
</tbody>
</table>

Other syntax conventions include the following:
• When you enter commands, separate parameters and keywords by at least one blank if there is no intervening punctuation.
• Footnotes are shown by a number in parentheses, for example, (1).
• Parameters with number values end with the symbol #.
• Parameters that are names end with 'name'.
• Parameters that can be generic end with the symbol *.

Example Syntax Diagram
Here is an example syntax diagram that describes the hello command.

According to the syntax diagram, these commands are all valid versions of the hello command:

- hello
- hello \textbackslash name
- hello \textbackslash name, \textbackslash name
- hello \textbackslash name, \textbackslash name, \textbackslash name
- hello, \textbackslash your\_greeting
- hello \textbackslash name, \textbackslash your\_greeting
- hello \textbackslash name, \textbackslash name, \textbackslash your\_greeting
- hello \textbackslash name, \textbackslash name, \textbackslash name, \textbackslash your\_greeting
The space before the name value is significant. If you do not code name, you must still code the comma before your_greeting.

How to Send Your Comments

Your feedback is important in helping us provide the most accurate and highest quality information. If you have any comments about this or any other IMS information, you can do one of the following:

- Go to the IMS home page at www.ibm.com/ims. There you will find an online feedback page where you can enter and submit comments.
- Send your comments by e-mail to imspubs@us.ibm.com. Be sure to include the title, the part number of the title, the version of IMS, and, if applicable, the specific location of the text you are commenting on (for example, a page number in the PDF or a heading in the Information Center).
Summary of Changes

Changes to the Current Edition of This Book for IMS Version 8

This edition, which is available in softcopy format only, includes technical and editorial changes.

The following information has been updated:

- “Usage of CQSPUT” on page 121.
- In Figure 26 on page 187, the reference to a TRPUNCH DD card was removed.

Changes to This Book for IMS Version 8

This edition, which is available in softcopy format only, includes technical and editorial changes.


The IMS Version 8: Common Queue Server Guide and Reference contains new technical information for Version 8, as well as editorial changes.

This book contains new and pre-existing information about CQS processes, functions, commands, and facilities.

New information on the following CQS enhancements is included:

- A new coupling facility list structure called a resource structure that contains shared resources
- One new request, CQSUPD, and five modified requests to access resources on a resource structure
- Structure alter support for the new resource structures
- Structure copy, a variant of structure rebuild, is supported for resource structures
- Structure Duplex, which provides resource structure backup. Structure Duplex requires z/OS Version 1 Release 2 or subsequent versions, releases, or modification levels.
- Structure Repopulation, a process by which CQS clients repopulate a failed resource structure
- Structure full monitoring support for queue structures and resource structures

Library Changes for IMS Version 8

Changes to the IMS Library for Version 8 include the addition of new titles, the elimination of one title, organizational changes, and accessibility enhancements. Changes are indicated by a vertical bar (|) to the left of the changed text.
New, Revised, and Eliminated Titles

The following list details major changes to the IMS Version 8 library:

- **IMS Version 8: Common Service Layer Guide and Reference**
  
  The library includes new information: **IMS Version 8: Common Service Layer Guide and Reference** (CSL). This information is available only in PDF and BookManager formats.

- The information formerly titled **IMS Version 7: Common Queue Server and Base Primitive Environment Guide and Reference** has been divided in the IMS Version 8 library:
  
  - **IMS Version 8: Base Primitive Environment Guide and Reference**
  - **IMS Version 8: Common Queue Server Guide and Reference**

- The information formerly titled **IMS Version 7: Installation Volume 1: Installation and Verification** is now titled **IMS Version 8: Installation Volume 1: Installation and Verification**. All installation information is now in the IMS Version 8 Program Directory.

- **IMS Version 8: Sample Operating Procedures**

  This information is no longer produced for the IMS library from IMS Version 8 and after.

- The information formerly titled **IMS Version 8: IMS Java™ User’s Guide** is now titled **IMS Version 8: IMS Java Guide and Reference**.

Organizational Changes

Organizational changes to the IMS Version 8 library include changes to:

- **IMS Version 8: Application Programming: Database Manager**

- **IMS Version 8: Application Programming: EXEC DLI Commands for CICS® and IMS**

- **IMS Version 8: Application Programming: Transaction Manager**

- **IMS Version 8: Messages and Codes, Volume 1**

- **IMS Version 8: Utilities Reference: Database and Transaction Manager**

  The section titled “DL/I Return and Reason Codes” has been moved from **IMS Version 8: Application Programming: Database Manager** to **IMS Version 8: Messages and Codes, Volume 1**.

  The section titled “DL/I Status Codes” will now only appear in **IMS Version 8: Messages and Codes, Volume 1**.

  The section titled “MFS Language Utility” has been renamed to “MFS Language Utility Control Statements” and has been moved from **IMS Version 8: Application Programming: Transaction Manager** to **IMS Version 8: Utilities Reference: Database and Transaction Manager**.

Deleted Information

OS/390 does not support the Virtual Fetch function any longer. Consequently, all information associated with Virtual Fetch has been deleted from the following IMS Version 8 information:

- **IMS Version 8: Administration Guide: System**

- **IMS Version 8: Failure Analysis Structure Tables (FAST) for Dump Analysis**
Accessibility Enhancements

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS products, including IMS, enable users to:

- Use assistive technologies such as screen-readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

User Assistive Technologies

Assistive technology products, such as screen readers, function with the user interfaces found in IMS. Consult the assistive technology documentation for specific information when using it to access these interfaces.

Accessible Information

Online information for IMS Version 8 is available in BookManager format, which is an accessible format. All BookManager functions can be accessed by using a keyboard or keyboard shortcut keys. BookManager also allows you to use screen readers and other assistive technologies. The BookManager READ/MVS product is included with the OS/390 base product, and the BookManager Library Reader™ (for workstations) is available on the IMS Licensed Product Kit (CD), which is available for downloading from IBM® at www.ibm.com

Keyboard Navigation of the User Interface

Users can access IMS user interfaces using TSO/E or ISPF. Refer to the z/OS: TSO/E Primer, z/OS: TSO/E User’s Guide, z/OS: ISPF User’s Guide. These guides describe how to navigate each interface, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.
Chapter 1. Introduction to the Common Queue Server

The Common Queue Server Guide and Reference is designed to help programmers, operators, and system support personnel perform these tasks:
- Plan for and design the installation of Common Queue Server (CQS)
- Install and operate CQS
- Diagnose and recover from CQS system problems
- Write a CQS client

In this chapter:
- “Terms Used in This Book”
- “Common Queue Server Overview” on page 4
- “Planning for CQS Hardware and Software Requirements” on page 9

This chapter contains General-Use Programming Interface information.

Terms Used in This Book

This book introduces several new terms or uses terms with a definition that is specific to CQS.

Definitions:

**Base Primitive Environment (BPE)**
A system-service layer component of IMS that provides a common set of system services. For more information about BPE, see IMS Version 8: Base Primitive Environment Guide and Reference.

**checkpoint data set**
A local data set that contains CQS system checkpoint information about a group of shared queues.

**client**
An OS/390 address space that uses CQS. For example, IMS and the Resource Manager (RM) are clients of CQS.

**cold queue**
A CQS private queue type that contains indoubt data objects for a client that cold started or a CQS that cold started.

**Common Queue Server**
A generalized server that manages objects on a coupling facility list structure, such as a queue structure or a resource structure.

**Common Service Layer (CSL)**
A group of address spaces in which one or more Common Service Layer components (Operations Manager, Resource Manager, or Structure Call Interface) are running. The Common Service Layer of an IMS system within a sysplex contains manager address spaces to control system management functions. The systems management services provided by the CSL are available to all IMSplex components. For more detailed information about CSL processes and functions, refer to the IMS Version 8: Common Service Layer Guide and Reference.

**coupling facility**
A special logical partition that provides high-speed caching, list processing, and...
locking functions in a sysplex. IMS saves global information in the coupling facility, so all the IMSs in the IMSplex have access to the global information.

CQS restart
Process by which CQS starts up; either a cold start or a warm start. During a CQS warm start the CQS environment is restored to the state it was in when CQS terminated. During a CQS cold start the CQS environment is not restored to a previous state; it is reinitialized.

data object
A piece of client data that is placed on a shared queue structure or a resource structure using a CQS request.

IMSplex
An IMSplex is one or more IMS address spaces (control, manager, or server) that work together as a unit.

list entry controls
Part of a coupling facility list structure list entry that contains control information associated with the list entry such as the entry key and entry ID.

list entry ID (LEID)
A unique identifier permanently assigned to each list entry in a coupling facility list structure by the system.

list header
Anchors the list to a coupling facility list structure and contains control information associated with the list.

list header number
The number of the list header, which identifies the particular list. The list header number ranges from 0 to the maximum defined by the first connector to the coupling facility list structure. The list header number must be specified on IXL requests that access the list structure, to identify the list to which the request applies.

list structure
A coupling facility structure that enables multisystem applications in a sysplex to share information organized as a set of lists or queues. A list structure consists of a set of lists and an optional lock table, which can be used for serializing resources in the list structure. Each list consists of a queue of list entries.

Local list entry (CQSLOCAL)
One CQSLOCAL is created by each CQS connected to the structure and contains data local to that CQS.

log token
A token that identifies a particular log record in the MVS™ log stream that is used to locate that log record.

master CQS
The CQS that coordinates a sysplex-wide task. The other CQSs sharing in the task are participants. If the master CQS fails for any reason, another CQS takes over the role of master and either continues or aborts the task.

overflow structure
A coupling facility list structure that contains shared queues when the primary structure reaches an installation-specified overflow threshold. The overflow structure is optional.
primary structure
An MVS coupling facility list structure that contains shared queues or shared resources.

queue
A collection of data objects with the same name in a Shared Queue.

queue name
A 16-byte name used to identify the queue that contains a data object on the shared queues. Multiple data objects may be identified by the same name.

queue structure
A coupling facility list structure, managed by CQS, which contains data objects grouped together in queues.

queue type
A grouping of shared queues on the coupling facility list structure whose meaning is client defined or CQS defined. A queue type is a queue structure attribute that allows CQS to physically group queues on a queue structure. CQS supports 11 private queue types for its own use and 11 client queue types for client use. The client-defined queue types have meaning to the client. A queue type can have a value from 1 to 255. Any queue type greater than 11 is mapped to one of the 11 physical queue types. An example of a CQS private queue type is the lock queue, in which a data object is moved when a CQSREAD request is issued. An example of a client queue type is the IMS transaction ready queue, containing transactions that are ready to be processed.

Resource Manager (RM)
An address space that provides resource services for IMSs in an IMSplex. It is part of the Common Service Layer of an IMSplex. IMS uses the Resource Manager to manage global resources and to coordinate online change of resources across an IMSplex.

resource name
An 11-byte unique name of a client-defined resource. An example of an IMS resource name is a transaction name.

resource structure
A coupling facility list structure, used by the Common Service Layer's Resource Manager and managed by CQS, that contains uniquely named resources. This structure is typically used to maintain global resource information when multiple Resource Managers exist in an IMSplex.

resource type
A CQS defined resource. CQS groups list headers into resource types. The resource types allow CQS and its clients to physically group resources of a particular type on a coupling facility list structure. A resource type is a resource structure attribute that allows CQS to physically group resources on a resource structure. CQS supports 11 private resource types for its own use and 11 client resource types for client use. The client-defined resource types have meaning to the client. A resource type can have a value of 1 to 255. Any resource type greater than 11 is mapped to one of the 11 physical resource types.

shared queue
A coupling facility structure, managed by CQS, that contains the data objects that are available to all CQS clients. All IMSs in an IMSplex have access to shared queues.

structure pair
A primary queue structure and its associated overflow structure.
structure recovery data set (SRDS)
Shared data sets that contain structure checkpoint information for shared queues on a structure pair. There are two SRDS data sets per structure pair.

Sysplex
A set of MVS systems communicating and cooperating with each other through certain multisystem hardware components and software services to process user workloads. Sysplex hardware includes coupling facilities, sysplex timers, System/390® processors. Sysplex software includes the MVS operating system and IMS.

unit of work (UOW)
To CQS, a client-defined grouping of data objects.

Common Queue Server Overview

Common Queue Server (CQS) is a generalized server that manages data objects on a coupling facility list structure, such as a queue structure or a resource structure, on behalf of multiple clients. CQS receives, maintains, and distributes data objects from shared queues on behalf of multiple clients. Each client has its own CQS access the data objects on the coupling facility list structure. IMS is one example of a CQS client that uses CQS to manage both its shared queues and shared resources.

Related Reading: See OS/390 MVS Setting Up a Sysplex, GC28-1779, for complete details about setting up a sysplex.

CQS runs on an OS/390 or z/OS operating system. The CQS client must also run under the same OS/390 or z/OS operating system. CQS runs in a separate address space that can be started by the client.

CQS uses the OS/390 or z/OS coupling facility as a repository for data objects. Storage in a coupling facility is divided into distinct objects called structures. Authorized programs use structures to implement data sharing and high-speed serialization. The coupling facility stores and arranges the data according to list structures. Queue structures contain collections of data objects that share the same name, known as queues. Resource structures contain data objects organized as uniquely named resources.

Clients communicate with CQS using CQS requests that are supported by CQS macro statements. Using these macros, CQS clients can communicate with CQS and manipulate client data on shared coupling facility structures. Figure 1 on page 5 shows the communications and the relationship between clients, CQSs, and the coupling facility.
CQS enables users to take advantage of the benefits of a Parallel Sysplex environment. These benefits include:

- **Automatic work load balancing**
  CQS places data objects on shared queues where they can be processed by any participating client system.
  Any participating client system can use CQS to retrieve a data object from the shared queues.

- **Incremental growth**
  Customers can add new systems as workload increases.

- **Reliability**
  For both shared queues and resources, if one client system fails, the remaining client systems process the work.

**CQS Components**

CQS maintains the following components:

- **Primary structure**
  An OS/390 coupling facility list structure that contains shared queues.

- **Resource structure**
  An OS/390 coupling facility list structure that contains uniquely named resources.

- **Overflow structure**
  A coupling facility list structure that contains shared queues when the primary structure reaches an installation-specified overflow threshold. The overflow structure is optional.

- **MVS log stream**
  A shared MVS log stream that contains all CQS log records from all CQSs connected to a structure pair. This log stream is important for recovery of shared queues, if necessary. Each structure pair has an associated log stream.

- **Checkpoint data set**
  A local data set that contains CQS system checkpoint information.
CQS Overview

- Structure recovery data sets (SRDS)
  Shared data sets that contain structure checkpoint information for shared queues on a structure pair. Each structure pair has two associated SRDSs.

CQS Functions

CQS provides the following functions:

- CQS requests
  An architected interface that clients use to access CQS or data objects on a queue structure or a resource structure.

- Notification of work on a queue
  Clients register interest in the shared queues. If an empty queue becomes non-empty, CQS notifies its registered clients.

- Records restart and recovery information
  CQS records all the information necessary for restart and recovery in the MVS system logger.

- CQS system checkpoint
  CQS system checkpoint writes log records relating to a particular CQS to the CQS log. The log records contain information necessary for CQS to restart and recover work.

- Structure checkpoint
  The structure checkpoint copies the queues from a structure pair into an SRDS for recovery purposes.

- Structure rebuild
  Structure rebuild is an OS/390 process that allows another instance of a structure to be allocated with the same name and data reconstructed from the initial structure instance.

- Overflow processing
  CQS provides an overflow option to help prevent a queue full condition. When the primary list structure reaches the overflow threshold value, CQS attempts to dynamically increase the size of the primary structure, offload selected queues to an overflow structure, or reject requests for selected queues.

Structures Managed by CQS

CQS can manage queue structures, resource structures, or both types of structures.

Queue Structures

A queue structure is a coupling facility list structure that contains collections of data objects with the same name. A collection of data objects is called a queue.

Queue structures support structure overflow, in which an associated overflow structure can be allocated to prevent the queue structure from becoming full. A primary queue structure and its associated overflow structure are known as a structure pair.

CQS physically divides the queue structure list headers into 11 private queue types for CQS use and 11 client queue types for client use. Client queue types are defined by the client. A client can group queues associated with a type of work, such as transactions. A queue type can have a value of 1 to 255. Any queue type over 11 is mapped into one of the physical queue types.
CQS manages private queues and client queues on queue structures. CQS uses the private queue types to manipulate client data objects for CQS requests. Each client queue type can be used by a client for a different type of work. A client registers interest in only those queue types that it can process, based on the types of work you define for it.

Five of the private queue types, and the work that a client processes on them, are shown in Table 2.

<table>
<thead>
<tr>
<th>Queue Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cold queue</td>
<td>Contains data objects that are in doubt for a client or for a CQS that cold started</td>
</tr>
<tr>
<td>Control queue</td>
<td>Contains control list entries that CQS uses to manage list structures and control processes (such as structure checkpoint and structure recovery)</td>
</tr>
<tr>
<td>Delete queue</td>
<td>Intermediate queue used for CQSDEL request processing</td>
</tr>
<tr>
<td>Lock queue</td>
<td>Contains data objects that are locked by the CQSREAD request</td>
</tr>
<tr>
<td>Move queue</td>
<td>Intermediate queue used for CQSMOVE request processing</td>
</tr>
</tbody>
</table>

**Resource Structures**

A resource structure is a coupling facility list structure, used by the Common Service Layer’s Resource Manager and managed by CQS, that contains uniquely named resources. This structure is typically used to maintain global resource information when multiple Resource Managers exist in an IMSplex. Resource structures enable CQS to perform resource management in an IMSplex.

CQS physically divides the resource structure list headers into 11 private resource types for CQS use and 11 client resource types for client use. Client resource types are defined by the client. A resource type can have a value of 1 to 255. Any resource type over 11 is mapped into one of the physical resource types.

Clients can use the resource structure to share resource information, control block information, and other types of information. The resource name is unique within the structure. Resources can be updated, queried, or deleted. A primary coupling facility list structure is used to contain the resources.

**CQS Structure Functions**

CQS provides functions for monitoring structure status and capacity, and enabling structure recovery. Some of these functions are built-in and do not require intervention. Other functions are optional, and can be set up or initiated as your installation needs them.

**Structure Overflow**

CQS provides a structure overflow function that automatically warns you when a queue structure is approaching full and takes action to prevent a full structure. When the usage of a structure reaches the overflow threshold, CQS attempts to make the structure larger by initiating a structure alter. If the alter fails, CQS either allocates an overflow structure and moves selected queues to the overflow structure (if you define an overflow structure), or prevents new data objects from being put on the selected queues.

**Important:** Overflow processing is not supported for resource structures.
CQS Overview

**Related Reading:** For detailed information about monitoring queue structure sizes and customizing CQS behavior in an overflow situation, see “Preventing CQS Structure Full” on page 41.

**Structure Rebuild**
Structure rebuild is an OS/390 process that allows another instance of a structure to be allocated with the same name and contain data reconstructed from the initial structure instance. OS/390 supports system-managed rebuild, in which case OS/390 rebuilds the structure. OS/390 also supports user-managed rebuild; the user rebuilds the structure.

CQS allows system-managed rebuild for queue structures and resource structures. CQS provides user-managed rebuild to support a structure copy function and a structure recovery function.

Structure copy copies the contents of a structure to another structure for planned reconfiguration. Structure copy is supported for resource structures and queue structures.

Structure recovery recovers a structure from the structure checkpoint data set and the CQS log after a structure failure. Structure recovery is supported for queue structures. Structure rebuild can be initiated manually by using an operator command, or automatically by CQS or OS/390.

**Related Reading:** For more information about rebuilding structures, see “Rebuilding Structures in CQS” on page 43.

**Structure Duplexing**
CQS can use the duplexing capabilities of z/OS Version 1 Release 2 or subsequent versions, releases, and modification levels. Duplexing occurs when the operating system creates a duplex (backup) copy of a structure, then maintains the two structures during normal mainline operation. If a structure fails, or a connection to a structure is lost, the operating system switches to the unaffected structure instance.

Structure duplexing requires z/OS Version 1 Release 2 or subsequent versions, releases, and modification levels.

**Related Reading:** Refer to Chapter 3, “CQS Administration,” on page 31 for more information about setting up and using structure duplexing.

**CQS Recovery Functions**
CQS provides functions for recovering work-in-progress, queues, and resources in case of system shutdown or failure. Some of these recovery functions are built-in and do not require intervention. Other functions are optional and can be set up or initiated as you need them.

**System Checkpoint**
To enable CQS restart in the event of failure, CQS periodically takes a “snapshot” of all control blocks and tables, and writes that information to the MVS log. That process is called system checkpoint. System checkpoint can be initiated by CQS, the client, or manually with an IMS command.

**Related Reading:** See “Using CQS System Checkpoint” on page 39 and “CQS Structure Event User-Supplied Exit Routine” on page 66 for detailed information about when system checkpoint occurs, the specific data that gets collected, and how that data is used during recovery.
CQS Logging and the MVS System Logger
CQS always uses the MVS system logger to record information necessary for CQS to recover queue structures and restart. CQS writes log records for each coupling facility list structure pair that it uses to a separate log stream. The log stream is shared among all CQS address spaces that share the structure. The system logger provides a merged log for all CQS address spaces that are sharing queues on a coupling facility list structure.

**Important:** Changes to resource structures are not logged.

**Related Reading:** For more information about logging, see “Recording Information Necessary for Starting CQS” on page 32.

Structure Checkpoint
To enable queue structure recovery in case of failure, CQS periodically takes a "snapshot" of the queues on all queue structures. That process is called structure checkpoint. Structure checkpoint can be initiated by CQS, the client, or manually with an IMS command.

**Important:** Structure checkpoint is not supported for resource structures.

**Related Reading:** See “Using CQS Structure Checkpoint” on page 39 and “CQS Structure Statistics User-Supplied Exit Routine” on page 57 for detailed information about when structure checkpoint occurs, what data gets collected, and how that data is used during recovery.

CQS Client Requests
CQS client systems communicate with CQS using a general use interface consisting of CQS requests. CQS requests are described in Chapter 6, “CQS Client Requests,” on page 85.

Planning for CQS Hardware and Software Requirements
Refer to the **IMS Version 8: Release Planning Guide** for complete information about the minimum hardware and software requirements, including operating system requirements, for setting up and running a CQS.

**Related Reading:** See System/390: Sysplex Hardware and Software Migration for more information on:
- The planning required to migrate to a sysplex that uses a coupling facility
- Hardware configurations of a sysplex
- The software products that can use the coupling facility
- The tasks for migrating to a coupling environment
- Checklists for installing the sysplex hardware and software
Planning for Hardware and Software Requirements
Chapter 2. CQS Definition and Tailoring

This chapter describes the tasks of defining and tailoring CQS. It provides detailed descriptions of macros, procedures, and other system-oriented information.

In this chapter:

- "CQS As Part of a Sysplex"
- "CQS and Defining MVS Policies"
- "CQS's Support of Multiple Clients" on page 14
- "Preparing to Start CQS" on page 15
- "Updating MVS Program Properties Table for CQS" on page 16
- "CQS Execution Parameters" on page 17
- "CQS Initialization Parameters PROCLIB Member (CQSIPxxx)" on page 19
- "CQS Local Structure Definition PROCLIB Member (CQSSLxxx)" on page 20
- "CQS Global Structure Definition PROCLIB Member (CQSSGxxx)" on page 22
- "CQS Execution Data Sets" on page 27
- "CQS System Checkpoint Data Set" on page 27
- "CQS Structure Recovery Data Sets" on page 28

This chapter contains Product-sensitive Programming Interface information.

CQS As Part of a Sysplex

An IMS sysplex typically consists of the following software, hardware, and z/OS or OS/390 policies:

- CQS for managing shared queues and resources
- CQS clients
- z/OS or OS/390 Operating System
- Signaling paths between systems
- Sysplex Couple Data Set that contains MVS information related to the sysplex
- Sysplex Failure Management (SFM) policy
- Automatic Restart Management (ARM) policy
- System Logger (LOGR) policy
- Coupling facility to contain CQS structures

The following section, "CQS and Defining MVS Policies," provides guidance on how to define the CFRM, SFM, and LOGR policies for a typical IMS sysplex using CQS. "CQS Execution Data Sets" on page 27 and "Using the MVS Automatic Restart Manager with CQS" on page 35 provide guidance on using the Automatic Restart Manager with CQS.

Related Reading: For detailed information about setting up and configuring a sysplex, refer to MVS Setting Up a Sysplex.

CQS and Defining MVS Policies

CQS is a component of IMS. Before you enable a CQS, however, you must define how the CQS is to use certain MVS services. The definitions are specified in policies.

© Copyright IBM Corp. 1997, 2005
Defining MVS Policies

Definition: A policy is a set of rules and actions that systems in a sysplex must follow when using certain MVS services. A policy allows MVS to manage specific resources in compliance with your system and resource requirements, but with little operator intervention. A policy can be set up to govern all systems in the sysplex or only selected systems. You might need to define more than one policy to allow for varying workloads, configurations, or other installation requirements at different times. For example, you might need to define one policy for your prime shift operations and another policy for other shifts. Although you can define more than one policy of each type (except for system logger) only one policy of each type can be active at a time. For system logger, only one LOGR policy is in the sysplex.

The following policies are used by MVS for systems management in a sysplex environment and are required for the CQS:

- The automatic restart management (ARM) policy defines how MVS is to manage restarts for specific MVS jobs and started tasks that are registered as elements of automatic restart management.

- The coupling facility resource management (CFRM) policy allows you to define how MVS manages coupling facility resources. One of the definitions in the CFRM policy is the coupling facility structure sizes. For more information on determining these sizes, see "Determining Structure Size for CQS Connections" on page 15.

- The system logger policy (LOGR) allows you to define, update, or delete structure or log stream definitions.

You must specify the MAXBUFSIZE parameter in the LOGR policy with a value that is large enough to contain the largest log record written by CQS.

Recommendation: Specify the MAXBUFSIZE parameter as 65272 bytes.

- The sysplex failure management (SFM) policy allows you to define responses for system failures, signalling-connectivity failures in the sysplex and reconfiguring systems in a Processor Resource/Systems Manager™ (PR/SM™) environment. The SFM policy is optional.

Figure 2 on page 13 shows an example of a CFRM policy that defines the following IMS resources:

- EMHQ primary structure
- EMHQ overflow structure
- EMHQ log structure
- MSGQ primary structure
- MSGQ overflow structure
- MSGQ log structure
- Resource structure

Figure 3 on page 14 shows an example of how the LOGR policy can be defined.

Figure 4 on page 14 shows an example of how the SFM policy can be defined.

Requirement: Run each policy that you create as a separate job. If you attempt to run all policies together as one job, the job will fail.

Related Reading: For information on defining and activating policies, see:

- *MVS Setting Up a Sysplex* for the CFRM, SFM, LOGR, and ARM policies
- *MVS Programming: Assembler Services Guide* for the LOGR policy
The example in Figure 2 shows you how to define IMS resources in the CFRM policy.

```
//CFRMPLCY JOB MSGCLASS=A,REGION=2000K,CLASS=K
// MSGLEVEL=(1,1)
//*****************************************************************************/
/* This JCL is used for configuration. INITSIZE is used for the primary MSGQ and EMHQ structures. */
//*****************************************************************************/
// 2 CF
//*****************************************************************************/
//POLICY EXEC PGM=IXCM2APU
//STEP LIB DD DSN=SYS1.MIGLIB,DISP=SHR
//SYS PRINT DD SYSOUT=A
//SYSIN DD *

DATA TYPE(CFRM)
DEFINE POLICY
   NAME(CONFIG01) REPLACE(YES)
   CF NAME (CF01) TYPE(nnnnnn) MFG(aa) PLANT(nn) SEQUENCE(nnnnnnnnnnn) PARTITION(n) CPCID(nn)
   CF NAME (CF02) TYPE(nnnnnn) MFG(aa) PLANT(nn) SEQUENCE(nnnnnnnnnnn) PARTITION(n) CPCID(nn)
   .
   .
   .
   STRUCTURE NAME(QMSGIMS01)
      SIZE(16000) INITSIZE(8000) PREFLIST(CF01,CF02) REBUILDPERCENT(1)
   STRUCTURE NAME(QMSGIMS01OFLW)
      SIZE(8000) PREFLIST(CF01,CF02) REBUILDPERCENT(1)
   STRUCTURE NAME(QEMHIMS01)
      SIZE(16000) INITSIZE(10000) PREFLIST(CF01,CF02) REBUILDPERCENT(1)
   STRUCTURE NAME(QEMHIMS01OFLW)
      SIZE(8000) PREFLIST(CF01,CF02) REBUILDPERCENT(1)
```

Figure 2. Defining IMS Resources in the CFRM Policy (Part 1 of 2)
Defining MVS Policies

```plaintext
STRUCTURE NAME(MVSLOGQMSG01)
  SIZE(16000)
  INITSIZE(11000)
  PREFLIST(CF01,CF02)

STRUCTURE NAME(MVSLOGQEMH01)
  SIZE(40000)
  PREFLIST(CF01, CF02)
  REBUILDPERCENT(1)

STRUCTURE NAME(QRSCIMS01)
  SIZE(16000)
  INITSIZE(8000)
  ALLOWAUTOALT(YES)
  FULLTHRESHOLD(60)
  DUPLEX(ALLOWED)
  PREFLIST(CF01,CF02)
  ...
  ...
```

Figure 2. Defining IMS Resources in the CFRM Policy (Part 2 of 2)

The example in Figure 3 shows you how to define IMS resources in the LOGR policy.

```plaintext
DATA TYPE(LOGR)
DEFINE STRUCTURE NAME(MVSLOGQMSG01)
  LOGSNUM(1)
  AVGBUF SIZE(4096)
  MAXBUFSIZE(65272)
DEFINE LOGSTREAM NAME(SYSLOG.QMSG01.LOG)
  STRUCTNAME(MVSLOGQMSG01)
  LS_MGMTCLAS(aaa)
  HLQ(IXGLOGR)  LS_SIZE(nnn)
```

Figure 3. Defining IMS Resources in the LOGR Policy

The example in Figure 4 shows you how to define IMS resources in the SFM policy.

```plaintext
DATA TYPE(SFM)
DEFINE POLICY
  NAME(SFMPOL)
  REPLACE(YES)
  CONNFAIL(YES)
  SYSTEM
    NAME (*)
    WEIGHT(10)
    ISOLATETIME(5)
  /*
```

Figure 4. Defining IMS Resources in the SFM Policy

CQS’s Support of Multiple Clients

You can use one CQS address space to support multiple clients. Examples of clients are IMS and RM.

As many as 32 different clients on the same MVS image can connect to coupling facility structures through a single CQS by using the CQSCONN request. For example, as many as 32 different IMS control regions can specify the same
CQS=xxx parameter in the DFSSQxx PROCLIB member. IMS starts the CQS address space if it is not currently active. If CQS is already active, IMS registers with the active CQS address space and does not start an additional CQS address space. Be sure that not more than 32 IMSs or RMs specify the same CQS.

### Determining Structure Size for CQS Connections

The size of the structures to which CQS connects is defined in the CFRM policy by defining the INITSIZE (initial size of the structure) and SIZE (maximum size of the structure) parameters. The initial size of a structure on the coupling facility is determined by the value of the INITSIZE parameter in the CFRM policy. When the first CQS connects to a structure, the size of that structure is the value specified for INITSIZE. If enough free space does not exist for this INITSIZE value, the size of the structure becomes that of the available space in the coupling facility.

To determine what structure size to define in the CFRM policy, you can use the S/390® Coupling Facility Structure Sizer Tool (CFSizer). CFSizer is a Web-based application that calculates the structure size based on the input data you provide. The CFSizer tool is available at: www.ibm.com/servers/eserver/zseries/cfsizer.

You can use structure alter to change the structure size or to redistribute the objects within the structure after it has been defined. For information, see: [Using Structure Alter for CQS](#) on page 38.

### Preparing to Start CQS

Because they are a part of IMS, CQS and BPE are automatically linked into IMS.SDFSRESL when you run the JCLIN jobstream.

Before you start CQS, complete the following tasks:

1. Create a coupling facility resource management (CFRM) policy that defines the structures to which you want CQS to connect. The CFRM policy specifies the name, size, attributes, and location that the structure is to be assigned when it is allocated.
2. Define the following MVS policies:
   - Sysplex failure management (SFM) policy – Optional
   - System logger (LOGR) policy
   - Automatic restart management (ARM) policy – Optional
3. Activate the CFRM policy using the following command:
   ```
   SETXCF START,POLICY,TYPE=CFRM,POLNAME=CONFIG01
   ```
   The structure is then allocated when the first CQS connects to it.
4. If you are using the SFM policy, activate it using the following command:
   ```
   SETXCF START,POLICY,TYPE=SFМ,POLNAME=SFMPOL
   ```
5. Create the CQS and BPE PROCLIB members. For information on creating BPE PROCLIB members, see *IMS Version 8 Base Primitive Guide and Reference*.
6. Define all CQS execution data sets.
7. Customize your CQS environment; determine which exits you want to use and then write the exits.
8. Authorize connections to CQS structures.
9. Update the MVS program properties table.
Preparing to Start CQS


The user must define parameters before the CQS address space is started. These parameters can be either:

- In the CQSIPxxx PROCLIB member
- CQS execution parameters

To customize and monitor your CQS environment, you can use any of the following user exit routines:

- CQS Initialization/Termination
- CQS Client Connection
- CQS Queue Overflow
- CQS Structure Statistics
- CQS Structure Event
- BPE Statistics

Related Reading:
- For more information on defining MVS policies, see "CQS and Defining MVS Policies" on page 11.
- For more information on the CQS initialization parameters, see "CQS Initialization Parameters PROCLIB Member (CQSIPxxx)" on page 19.
- For more information on the CQS user exit routines, see Chapter 4, “CQS User-Supplied Exit Routines,” on page 51.
- For more information on BPE user exit routines, see the IMS Version 8: Base Primitive Environment Guide and Reference.

Updating MVS Program Properties Table for CQS

You must add an entry in the MVS program properties table (PPT) for CQS. The steps for doing this are:

1. Edit the SCHEDxx member of the SYS1.PARMLIB data set.
2. Add the entry shown in Figure 5 to the SCHEDxx member:

```plaintext
PPT PGNAME(CQSINIT0) /* PROGRAM NAME = CQSINIT0 */
   CANCEL /* PROGRAM CAN BE CANCELED */
   KEY(7) /* PROTECT KEY ASSIGNED IS 7 */
   NOSWAP /* PROGRAM IS NON-SWAPPABLE */
   NOPRIV /* PROGRAM IS NOT PRIVILEGED */
   DSI /* REQUIRES DATA SET INTEGRITY */
   PASS /* CANNOT BYPASS PASSWORD PROTECTION */
   SYST /* PROGRAM IS A SYSTEM TASK */
   AFF(NONE) /* NO CPU AFFINITY */
   NOPREF /* NO PREFERRED STORAGE FRAMES */
```

Figure 5. Entry to Be Added to the MVS Program Properties Table

3. Take one of the following actions to make the SCHEDxx changes effective:
   - Re-IPL the MVS system.
   - Issue the MVS SET SCH= command.

Related Reading: For additional reading about updating the program properties table, see MVS Initialization and Tuning Reference.
You can specify the following execution parameters on the CQS startup procedure. Read the descriptions of the parameters to determine whether you want to accept the system defaults or to tailor the system to fit the requirements of your environment.

**ARMRST= Y | N**

Specifies whether the MVS Automatic Restart Manager (ARM) is to be used to restart the CQS address space after an abend. If you specify Y (yes), ARM restarts the CQS address space after most system failures. If you specify N (no), ARM does not restart the CQS address space after any system failure.

ARM does not restart the CQS address space if the CQS abends before restart is complete.

To restart CQS when it has been cancelled by MVS, you must specify the ARMRESTART option of either the MVS CANCEL or FORCE command.

**Related Reading:** For information on the CANCEL and FORCE commands, see *MVS System Commands*.

If you specify this optional parameter, it overrides the value you specified in the CQSIPxxx PROCLIB member.

**BPECFG=**

Specifies the eight-character name of the BPE configuration PROCLIB member. You can specify this parameter only as an execution parameter.

**CQSGROUP=**

Specifies a one- to five-character identifier. CQS concatenates this identifier to the characters CQS to create the XCF CQS group name. You must use the same identifier for all CQS address spaces that share the same set of structures. You can also use the same identifier for the SQGROUP= parameter in the DFSSQxxx PROCLIB member.

If you specify this optional parameter, it overrides the value specified in the CQSIPxxx PROCLIB member. You must specify this parameter either as an execution parameter or in the CQSIPxxx PROCLIB member.

**CQSINIT=**

Specifies the three-character suffix for the CQS initialization parameters PROCLIB member, CQSIPxxx. You can specify this parameter only as an execution parameter. The default suffix is 000.

**SSN=**

Specifies the name for the CQS address space. The value must be one to four alphanumeric characters. If you specify this optional parameter, it overrides the value specified in the CQSIPxxx PROCLIB member. You must specify this parameter either as an execution parameter or in the CQSIPxxx PROCLIB member. This name is also used to create the CQSID, which is used in CQS processing. The eight-character CQSID is the SSN followed by the characters CQS.

**Example:** If SSN=ABC, CQSID=ABCCQS.

Trailing blanks are deleted and the CQSID is padded with blanks.

**STRDEFG=**

Specifies a three-character suffix for the CQS global structure definition PROCLIB member, CQSSGxxx. This member contains the parameters related to the coupling facility structures that are common to all CQS address spaces.
CQS Execution Parameters

that are sharing the queues. If you specify this optional parameter, it overrides the value specified in the CQSIPxxx PROCLIB member. The default suffix is 000.

**STRDEFL=**

Specifies a three-character suffix for the CQS local structure definition PROCLIB member, CQSSLxxx. This member contains the parameters that are related to the coupling facility structures and that are unique to an individual CQS address space. If you specify this optional parameter, it overrides the value specified in the CQSIPxxx PROCLIB member. The default suffix is 000.

**Related Reading:** See information in *IMS Version 8: Installation Volume 2: System Definition and Tailoring* to see the relationship of IMS PROCLIB members and CQS PROCLIB members.

Figure 6 shows the general relationship between execution parameters and PROCLIB members. In the figure, CQSSL001 and CQSSL002 contain structure definition parameters that are unique to each CQS. CQSSG00A contains the structure definition parameters that are shared by all CQS address spaces connected to the shared queues.

![Figure 6. Specifying IMS and CQS Parameters](image-url)
CQS Initialization Parameters PROCLIB Member (CQSIPxxx)

Use the CQSIPxxx PROCLIB member to specify parameters that are related to initialization of the CQS address space. You can use CQS execution parameters to override certain parameters within CQSIPxxx.

The following rules apply to the format of the CQSIPxxx member:

- The execution member consists of one or more fixed-length character records. (The configuration data set can be of any LRECL greater than eight, but it must be of fixed-record format.)
- The rightmost eight columns of each record are ignored and you can use them for sequence numbers or any other notation. In the remaining columns, you code the keyword parameters. For example, if your record size is 80, you use columns 1 through 72 for your configuration data. You can use columns 73 through 80 for sequence numbers.
- Keywords can contain leading and trailing blanks.
- Each record can contain multiple keywords.
- Use commas or spaces to delimit keywords.
- Use an asterisk (*) or pound sign (#) in column one to begin a comment. You can include a comment anywhere within a statement by enclosing it between a slash-asterisk and an asterisk-slash pair.
- Values coded in this PROCLIB member are case-sensitive.

A sample CQSIPxxx PROCLIB member is shown in Figure 7:

```
* CQS INITIALIZATION PROCLIB MEMBER *
**********************************************************************
ARMRST=Y /* ARM SHOULD RESTART CQS ON FAILURE */
CQSGROUP=GRUP1 /* GROUP NAME (XCF GROUP = GRUP1CQS) */
SSN=CQS1 /* CQS ADDRESS SPACE (CQSID = CQS1CQS) */
STRDEFG=190 /* GLOBAL STR DEFINITION MEMBER = CQSSG190 */
STRDEFL=191 /* LOCAL STR DEFINITION MEMBER = CQSSL191 */
IMSPLEX(NAME=PLEX1) /* IMSPLEX NAME(CSLPLEX1) */
**********************************************************************
```

Figure 7. Sample CQSIPxxx PROCLIB Member

**ARMRST= Y | N**

Specifies whether the MVS Automatic Restart Manager (ARM) is used to restart the CQS address space after an abend. If you specify Y (yes), ARM restarts the CQS address space after most system failures. If you specify N (no), ARM does not restart the CQS address space after any system failure.

ARM does not restart the CQS address space if the CQS abends before restart is complete.

To restart the CQS when it has been cancelled by MVS, you must specify the ARMRESTART option of either the MVS CANCEL or FORCE command.

**Related Reading:** For information on the CANCEL and FORCE commands, see MVS System Commands.

This parameter can be specified as an execution parameter on the CQS procedure to override the value in CQSIPxxx.
CQSGROUP=
Specifies a one- to five-character identifier. CQS concatenates this identifier to
the characters CQS to create the group name of the XCF CQS shared queues.
You must use the same identifier for all CQS address spaces that share the
same set of structures. You can also use the same identifier for the
SQGROUP= parameter in the DFSSQxxx PROCLIB member.
This parameter can be specified as an execution parameter on the CQS
procedure to override the value in CQSIPxxx.

IMSPLEX()
Specifies the IMSplex to which CQS joins. IMSPLEX is an optional parameter.
IMSPLEX does not have a default value. Only one IMSPLEX keyword can be
specified. The IMSPLEX definition parameter follows:

NAME=
A one- to five-character identifier that specifies the XCF CSL IMSplex group
name. CQS concatenates this identifier to CSL to create the XCF CSL
IMSplex group name. All OM, RM, SCI, IMS, CQS and similar address
spaces must specify the same to be part of the same IMSplex. The same
identifier must also be used for the IMSPLEX= parameter in the CSLStxxx,
CSLOIxxx, CSLRlxxx and DFSCGxxx PROCLIB members.

SSN=
Specifies the one- to four-character name for the CQS address space. The
name value must consist of alphabetic characters. You must specify this
parameter either as an execution parameter or in the CQSIPxxx PROCLIB
member. This name is also used to create the CQSID, which is used in CQS
processing. The eight-character CQSID is the SSN followed by the characters
CQS.

Example: If SSN=ABC, CQSID=ABCCQS.
Trailing blanks are deleted and the CQSID is padded with blanks.

STRDEFG=
Specifies a three-character suffix for the CQS global structure definition
PROCLIB member, CQSSGxxx. This member contains the parameters related
to the coupling facility structures that are common to all CQS address spaces
that are sharing the queues. If you specify this optional parameter, it overrides
the value specified in the CQSIPxxx PROCLIB member. The default suffix is
000.

STRDEFL=
Specifies a three-character suffix for the CQS local structure definition
PROCLIB member, CQSSLxxx. This member contains the parameters that are
related to the coupling facility structures and that are unique to an individual
CQS address space. If you specify this optional parameter, it overrides the
value specified in the CQSIPxxx PROCLIB member. The default suffix is 000.

CQS Local Structure Definition PROCLIB Member (CQSSLxxx)
Use the CQSSLxxx PROCLIB member to define local CQS parameters that are
related to one or more coupling facility structures. Each CQS should point to a
different CQSSLxxx member. CQS connects to each defined structure in the
member. The structures defined in the CQSSLxxx member must also be defined in
the CQSSGxxx PROCLIB member.
Important: The CQSSLxxx PROCLIB member applies to queue structures only, not resource structures. If you do not define queue structures, you do not need to define the CQSSLxxx PROCLIB member.

The following rules apply to the format of the CQSSLxxx member:

- The execution member consists of one or more fixed-length character records. (The configuration data set can be of any LRECL greater than eight, but it must be of fixed-record format.)
- The rightmost eight columns of each record are ignored and can be used for sequence numbers or any other notation. In the remaining columns, you code the keyword parameters. For example, if your record size is 80, you use columns 1 through 72 for your configuration data. You can use columns 73 through 80 for sequence numbers.
- Keywords can contain leading and trailing blanks.
- Each record can contain multiple keywords.
- Commas or spaces delimit keywords.
- A comment begins with an asterisk (*) or pound sign (#) in column one. You can include a comment anywhere within a statement by enclosing it between a slash-asterisk and an asterisk-slash pair.

```
/*This is an example of a comment within a statement*/
```

- Values coded in this PROCLIB member are case-sensitive.

A sample CQSSLxxx PROCLIB member is shown in Figure 8:

```
**********************************************************************
* LOCAL STRUCTURE DEFINITION PROCLIB MEMBER                      *
**********************************************************************

* DEFINITION FOR IMS MESSAGE QUEUE STRUCTURE                     *
STRUCTURE (STRNAME=QMSGIMS01, CHKPTDSN=CQSA.QMSG.IMS01.CHKPT, SYSCHKPT=50000)

* DEFINITION FOR IMS EMH QUEUE STRUCTURE                         *
STRUCTURE (STRNAME=QEMHIMS01, CHKPTDSN=CQSA.QEMH.IMS01.CHKPT, SYSCHKPT=50000)
```

Figure 8. Sample CQSSLxxx PROCLIB Member

Use the following keyword parameters to define a structure to CQS. The structure definition parameters must be enclosed within parentheses. The STRUCTURE keyword must precede the left parenthesis.

Example: STRUCTURE (STRNAME=strname, CHKPTDSN=chkptdsn, ...)

STRNAME=
The required 1- to 16-character name of the primary coupling facility structure to which CQS connects.

The installation must have defined the structure in the coupling facility resource management (CFRM) administrative policy. The structure name must follow the naming rules of the CFRM. If the name has fewer than 16 characters, CQS
pads the name with blanks. The valid characters are A-Z, 0-9, and the characters $, &, # and _. Names must be uppercase and start with an alphabetic character.

**Restriction:** Avoid using names IBM uses for its structures. Do not begin structure names with the letters A-I, or the character string SYS.

**CHKPTDSN=**

The required 1- to 44-character data set name of the cataloged VSAM data set that is used for the checkpoint data set for the indicated structure. The data set is dynamically allocated by CQS during CQS initialization. Each structure defined in CQSSLxxx must have a unique CHKPTDSN.

**SYSCHKPT=**

Specifies the number of log records CQS writes between system checkpoints. This value can be from 200 to 2147483647. Each CQS address space that is connected to a shared queue structure can specify a different system checkpoint log record count. This value is not shared between CQS address spaces.

This parameter has no default. If you do not specify a value, automatic system checkpoints are only taken during restart, normal shutdown, and after a structure checkpoint.

---

**CQS Global Structure Definition PROCLIB Member (CQSSGxxx)**

The CQSSGxxx PROCLIB member defines global CQS parameters that are related to one or more coupling facility structures. These parameters are shared by all CQS address spaces that share the structures. A particular CQS can support message queues, resource structures, or a combination of both message queues and resource structures. Each CQS sharing a structure must point to a CQSSGxxx member containing identical structure definition parameters.

**Recommendations:**

- Point all CQSs to the same CQSSGxxx member to avoid parameter mismatches. CQS connects to each structure that is defined in the member. The structures defined in the CQSSGxxx member must also be defined in the CQSSLxxx PROCLIB member.

- If you are using message queue structures, define an overflow structure name OVFLWSTR= if there is a possibility that you will use an overflow structure. If you have to add an overflow structure later, the structure and all CQSs must be cold started.

The following rules apply to the format of the CQSSGxxx member:

- The execution member consists of one or more fixed-length character records. (The configuration data set can be of any LRECL greater than eight, but it must be of fixed-record format.)

- The rightmost eight columns of each record are ignored and can be used for sequence numbers or any other notation. In the remaining columns, you code the keyword parameters. For example, if your record size is 80, you use columns 1 through 72 for your configuration data. You can use columns 73 through 80 for sequence numbers.

- Keywords can contain leading and trailing blanks.

- Each record can contain multiple keywords.

- Commas or spaces delimit keywords.
A comment begins with an asterisk (*) or pound sign (#) in column one. You can include a comment anywhere within a statement by enclosing it between a slash-asterisk and an asterisk-slash pair.

/*This is an example of a comment within a statement*/

Values coded in this PROCLIB member are case-sensitive.

A sample CQSSGxxx PROCLIB member that defines both message queue and resource structures is shown in Figure 9:

```
******************************************************
* GLOBAL_STRUCTURE DEFINITION PROCLIB MEMBER         *
******************************************************
*-------------------------------------------------*
* DEFINITION FOR IMS MESSAGE QUEUE STRUCTURES      *
*-------------------------------------------------*
STRUCTURE (STRNAME=QMSGIMS01,
            OVFLWSTR=QMSGIMS01OFLW,
            SRDSDSN1=CQS.QMSG.IMS01.SRDS1,
            SRDSDSN2=CQS.QMSG.IMS01.SRDS2,
            LOGNAME=SYSLOG.QMSG01.LOG
            OBJAVGSZ=1024)
*-------------------------------------------------*
* DEFINITION FOR IMS EMH QUEUE STRUCTURES           *
*-------------------------------------------------*
STRUCTURE (STRNAME=QEMHIMS01,
            OVFLWSTR=QEMHIMS01OFLW,
            SRDSDSN1=CQS.QEMH.IMS01.SRDS1,
            SRDSDSN2=CQS.QEMH.IMS01.SRDS2,
            LOGNAME=SYSLOG.QEMH01.LOG
            OBJAVGSZ=1024)
*-------------------------------------------------*
* DEFINITION FOR IMS RESOURCE STRUCTURE            *
*-------------------------------------------------*
RSRCSTRUCTURE (STRNAME=QRSCIMS01)
```

Figure 9. Sample CQSSGxxx PROCLIB Member

Use the following keywords to define a structure to CQS. At least one STRUCTURE or RSRCSTRUCTURE definition is required.

**STRUCTURE=**

Defines a queue structure to CQS. This keyword can be repeated. Keyword parameters must be enclosed within parentheses.

**Example:** STRUCTURE (STRNAME=string, SRDSDSN1=srsdsn1, ...)

The following keyword parameters are available to the STRUCTURE definition:

**STRNAME=**

The required 1- to 16-character name of the primary coupling facility structure to which CQS connects.

The installation must have defined the structure name in the CFRM administrative policy. The structure name must follow the naming rules of the CFRM. For names with fewer than 16 characters, CQS pads the name with blanks. The valid characters are A-Z, 0-9, and the characters $, &, # and _. Names must be uppercase and start with an alphabetic character.
**Restriction:** Avoid using names IBM uses for its structures. Do not begin structure names with the letters A-I, or with the character string SYS.

**SRDSDSN1=**
Is a required 1- to 44-character data set name of the cataloged VSAM data set that is used for the first structure recovery data set. The data set name is used to dynamically allocate the data set when a structure checkpoint is requested. For a given structure checkpoint request, CQS uses either structure recovery data set 1 or data set 2. CQS alternates between the two data sets for structure checkpoint processing.

All CQS address spaces that connect to a shared queue structure must use the same value for this parameter. The value specified by the CQS that initially allocates the structure is the value that is used for the life of the structure.

**SRDSDSN2=**
Is a required 1- to 44-character data set name for the cataloged VSAM data set that is used for the second structure recovery data set. The data set name is used to dynamically allocate the data set when a structure checkpoint is requested. For a given structure checkpoint request, CQS uses either structure recovery data set 1 or data set 2. CQS alternates between the two data sets for structure checkpoint processing.

All CQS address spaces that connect to a shared queue structure must use the same value for this parameter. The value specified by the CQS that initially allocates the structure is the value that is used for the life of the structure.

**LOGNAME=**
Is the required 1- to 26-character name of the MVS log stream that CQS uses to record all information related to the structure. The installation must have previously defined this name to the MVS system logger.

All CQS address spaces that connect to a shared queue structure must use the same value for this parameter. The value specified by the CQS that initially allocates the structure is the value that is used for the life of the structure.

**OBJAVGSZ=**
Specifies the average size of a data object that is written to a queue on this structure. This value can range from 128 bytes to 61 312 bytes or from 1 KB to 59 KB. The following list defines some IMS object sizes:

**IMS client**
The object size is the size of the IMS message plus some control information.

**IMS queue manager messages**
If the user message and the message queue prefix both fit completely into one queue buffer, the object size is the sum of the user message and the message queue prefix. If both parts do not completely fit into one queue buffer, the object size is the size of the portion of the message and the message queue prefix that do fit into one queue buffer. The size of an IMS message queue buffer is specified to the IMS control region by the QBUFSZ execution parameter.

**IMS expedited message handler messages**
The object size is the size of the user message plus 240 bytes (the size of the EMHB global header).
**Recommendation:** Specify the OBJAVGSZ to be the average of the sizes of all the objects passed to CQS by a CQSPUT request. CQS adds its own prefix containing control information to every object placed on the structure. CQS adds the length of its prefix to the OBJAVGSZ value that you specify to get the true average object size. Therefore, OBJAVGSZ should reflect only the average size of the objects as they are passed to CQS, not the average size of the object on the coupling facility.

If the OBJAVGSZ is too small, too much space in the structure is allocated for control information. The structure becomes full when all of the space for data is used up, even though space for control information is still available.

If the OBJAVGSZ is too large, too much space in the structure is allocated for data. The structure becomes full when all of the control space is used up, even though space for data is still available.

**Example:** Five objects are put on the structure by a CQSPUT request. The sizes of the objects are:

- object 1: 134 bytes
- object 2: 1066 bytes
- object 3: 3200 bytes
- object 4: 172 bytes
- object 5: 345 bytes

The average object size is calculated to be 983 bytes.

\[
\frac{134 + 1066 + 3200 + 172 + 345}{5} = 983
\]

**OVFLWMAX=**

Specifies the maximum threshold percentage for overflow processing. This value indicates the percentage of the structure that must be in use before CQS goes into overflow mode. This value can be from 50 to 100. For example, if OVFLWMAX=75, the structure is put into overflow mode when the structure usage reaches 75% of the structure size. The default is 70%.

The value specified by the CQS that initially allocates the structure is used for the life of the structure.

**OVFLWSTR=**

Is the 1- to 16-character name of the optional coupling facility structure to which CQS connects for structure overflow processing. The name must follow the same naming convention as the structure name specified by the STRNAME= parameter. When CQS is processing in overflow mode, selected queues are written to this structure instead of to the primary structure.

If an overflow structure is not specified and an overflow condition is detected, CQS rejects requests to add data objects to those queues that were selected for overflow.

If an overflow structure is specified, CQS connects to the overflow structure during CQS initialization and then again during phase one of overflow threshold processing. If CQS detects that the overflow structure size is less than 30% of the primary structure size, the overflow structure is considered to be too small and CQS issues the CQS0268I message. CQS is allowed to initialize even though the overflow structure is too small. CQS disconnects from and deletes the overflow structure at the end of CQS initialization.
CQS does not attempt to connect to the overflow structure again until the overflow threshold is reached. If at that time the overflow structure size is still less than 30% of the primary structure size, CQS again issues the CQS0268I message. CQS goes into overflow mode, but the overflow structure is not used. Requests to add data objects to those queues that were selected for overflow are rejected.

**Recommendation:** Define the size of the overflow structure in the CFRM policy to be at least X% of the primary structure size, where X is the value specified for the OVFLWMAX= parameter. The value specified for the OVFLWMAX= parameter indicates the percentage of the primary structure that must be in use before CQS goes into overflow mode, the overflow threshold. For example, if the overflow threshold was defined with the OVFLWMAX= parameter to be 75% of the primary structure size, the size of the overflow structure should be at least 75% of the primary structure size. If a value is not specified for the OVFLWMAX= parameter, the overflow threshold defaults to 70% and the size of the overflow structure should be at least 70% of the primary structure size.

An overflow structure name can be defined only when the structure is cold started. Once structures have been allocated, an overflow structure cannot be added unless the structure and all CQSs are cold started.

All CQS address spaces that connect to a shared queue structure must use the same value for this parameter. The value specified by the CQS that initially allocates the structure is used for the life of the structure.

**STRMIN=**

Specifies the value for the minimum primary structure size to which CQS can connect. This value is specified in units of 4 KB blocks and can be any value from 0 to the maximum structure size of 524288 (a 2-GB structure).

The default value is 0, indicating that CQS accepts the size as allocated by the coupling facility. If the coupling facility is constrained, the structure can be allocated to something smaller than that defined by the CFRM policy. Depending on the size, the message queues might overflow sooner than expected.

**Recommendation:** Specify a value for STRMIN= that is less than the structure size that is defined in the policy.

The value specified by the CQS that initially allocates the structure is used for the life of the structure.

When the first CQS connects to an empty structure, that structure is allocated on the coupling facility. After it is allocated, the structure remains on the coupling facility regardless of whether a CQS is connected to it.

If, during connection to a structure, CQS determines that the size of the structure is smaller than the minimum size and the structure is empty, CQS terminates. In this case, the installation needs to redefine the use of the coupling facility to ensure that the required size can be allocated. If CQS connects to a structure that is smaller than the minimum size, but the structure contains data objects, CQS does not terminate. CQS attempts to use the smaller structure because it already contains data. In this case, CQS issues a message that allows an operator to initiate a structure rebuild in order to increase the structure size.

**RSRCSTRUCTURE=**

Defines a resource structure to CQS. An IMSplex can define only one resource
structure; name uniqueness is within one resource structure. Keyword parameters must be enclosed within parentheses.

**Example:** RSRCSTRUCTURE (STRNAME=strname)

The following keyword parameter is available:

**STRNAME=**

The required 1- to 16-character name of the coupling facility list structure to which CQS connects. This parameter defines the name of the resource structure used by RM to keep IMS resource information.

The installation must have defined the structure name in the CFRM administrative policy. The structure name must follow the naming rules of the CFRM. For names with fewer than 16 characters, CQS pads the name with blanks. The valid characters are A-Z, 0-9, and the characters $, &, # and _. Names must be uppercase and start with an alphabetic character.

**Restriction:** Avoid using names IBM uses for its structures. Do not begin structure names with the letters A-I, or with the character string SYS.

---

**CQS Execution Data Sets**

CQS uses two types of data sets, the system checkpoint data set and the structure recovery data set. Both of these data sets must be VSAM entry-sequenced data sets (ESDSs). These data sets are user data sets (and are not known to SMP/E).

**CQS System Checkpoint Data Set**

Each CQS address space that is connected to a shared queues structure maintains a system checkpoint data set for each structure pair. Neither CQS address spaces nor shared queues structures share the system checkpoint data set. The CQS initialization process dynamically allocates the system checkpoint data set.

Use the MVS/DFP™ DEFINE CLUSTER functional command to define the data set to the installation.

**Related Reading:** For a description of the DEFINE CLUSTER function command and its parameters, see DFSMS Access Method Services for Catalogs.

**Requirement:** When you use the DEFINE CLUSTER functional command to define the system checkpoint data set, you must specify the following parameters:

**NAME**

Specifies the same name that you specify using the CHKPTDSN= parameter of the CQS local structure definition PROCLIB member.

**NONINDEXED**

Specifies that the data set is to be an ESDS.

**NONSPANNED**

Specifies that the records must be contained in a single control interval.

**SHAREOPTIONS**

Specifies how a cluster can be shared among users.

**Requirement:** You must specify SHAREOPTIONS (2,3).

**REUSE**

Specifies that the cluster can be reused.
Recommendation: The following parameters are recommended when you specify the DEFINE CLUSTER functional command:

**RECORDSIZE**  
Specifies the average and maximum length in bytes for the records in the cluster.

**Recommendation:** Both the maximum and minimum length of the RECORDSIZE should be 7 bytes less than the CONTROLINTERVALSIZE.

**CONTROLINTERVALSIZE**  
Specifies the size of the control interval for the cluster.

**Requirement:** The recommended CONTROLINTERVALSIZE is 512; it must be a multiple of 512.

**Example:** An example of the system checkpoint data set is shown in Figure 10.

```  
DEFINE CLUSTER          -  
    (NAME (MSGQ.CHKPT)) -  
    TRK(2,2) VOL (IMSQAV) NONINDEXED SHAREOPTIONS (2,3) -  
    RECSZ(505,505) REUSE CISZ (512))
```

*Figure 10. System Checkpoint Data Set Example*

**CQS Structure Recovery Data Sets**

CQS uses two structure recovery data sets per structure pair for its structure checkpoint processing.

Structure recovery data sets are not used for the resource structure.

When a structure checkpoint is requested, CQS dynamically allocates the structure recovery data sets. Structure checkpoint requests alternate between the two structure recovery data sets. During structure checkpoint processing all recoverable data objects on a structure are written to the structure recovery data sets. Thus, the size of each data set should be approximately the size of the primary structure plus the overflow structure to ensure that the entire structure fits in the data set.

Use the MVS/DFP DEFINE CLUSTER functional command to define the data set to the installation.

**Related Reading:** For a description of the DEFINE CLUSTER function command and its parameters, see *DFSMS Access Method Services for Catalogs*.

**Requirement:** When you use the DEFINE CLUSTER functional command to define the structure recovery data set, you must specify the following parameters.

**NAME**  
Specify the same name you specify in the CQS global structure definition PROCLIB member using the SRSDDSN1= and the SRSDDSN2= parameters.

**NONINDEXED**  
Specifies that the data set is to be an ESDS.

**NONSPANNED**  
Specifies that the records must be contained in a single control interval.
SHAREOPTIONS
Specifies how a cluster can be shared among users.

Requirement: You must specify SHAREOPTIONS (2,3).

REUSE
Specifies that the cluster can be reused.

Recommendation: The following parameters are recommended when you specify the DEFINE CLUSTER functional command:

RECORDSIZE
Specifies the average and maximum length in bytes for the records in the cluster.

Recommendation: Both the maximum and minimum length of the RECORDSIZE should be 7 bytes less than the CONTROLINTERVALSIZE.

CONTROLINTERVALSIZE
Specifies the size of the control interval for the cluster.

Requirement: The recommended control interval size is 32768; it must be a multiple of 512.

Example: An example of the structure recovery data set is shown in Figure 11:

```
DEFINE CLUSTER
  (NAME (MSGQ.SRDS1) -
   TRK(45,5) VOL (DSHR03) NONINDEXED SHAREOPTIONS (2,3) -
   RECSZ(32761,32761) REUSE CISZ (32768)
```

Figure 11. Structure Recovery Data Set Example
Execution Data Sets
Chapter 3. CQS Administration

This chapter describes the system administration tasks associated with using the Common Queue Server.

In this chapter:

- "Starting CQS"
- "Recording Information Necessary for Starting CQS" on page 32
- "Restarting CQS" on page 34
- "Restarting CQS Structures" on page 32
- "Establishing Client Connection to CQS During Failed Client Takeover" on page 36
- "Authorizing Access To CQS" on page 37
- "Using Structure Alter for CQS" on page 38
- "Using CQS System Checkpoint" on page 39
- "Using CQS Structure Checkpoint" on page 39
- "Preventing CQS Structure Full" on page 41
- "Rebuilding Structures in CQS" on page 43
- "Deleting a Structure When CQS Is Not Connected" on page 48
- "Shutting Down CQS" on page 48

This chapter contains Product-sensitive Programming Interface information.

Starting CQS

You can activate CQS in one of three ways:
- As an MVS task, using the MVS START command
- As an MVS batch job
- As a client task — Some clients (such as IMS) automatically start CQS, when appropriate

**Example:** If IMS is the client, define the CQS name in the DFSSQxxx PROCLIB member and specify the CQS name on the SHAREDQ parameter of the IMS procedure. IMS doesn’t start CQS if the CQS is needed only to manage a resource structure.

A CQS that supports only a resource structure must be started manually because IMS does not start this CQS.

**Related Reading:**
- For information on defining IMS procedures, see DCC Procedures and DFSSQxxx PROCLIB member in *IMS Version 8: Installation Volume 2: System Definition and Tailoring*
- For information on defining MVS policies, see "CQS and Defining MVS Policies" on page 11.
- For information on the CQS initialization parameters, see "CQS Initialization Parameters PROCLIB Member (CQSIPxxx)" on page 19.
Recording Information Necessary for Starting CQS

CQS uses the MVS system logger to record all information necessary for CQS to recover queue structures and restart. CQS writes log records for each coupling facility list structure pair that it uses to a separate log stream. The log stream is shared among all CQS address spaces that share the queue structure. The MVS system logger provides a merged log for all CQS address spaces that are sharing queues on a coupling facility list structure.

**Important:** Changes to resource structures are not logged.

For CQS to use an MVS system log, you must first define the log stream and associated resources to MVS.

CQS also provides a File Select and Formatting Print utility to print the log records.

**Related Reading:** For more information on defining the log stream, see *MVS Setting Up a Sysplex*.

For more information on the File Select and Formatting Print utility, see "Printing CQS Log Records" on page 187.

Restarting CQS Structures

Restarting structures involves restarting a structure pair. The structure restart function ensures that the data in the structure is correct. Structure restart deals only with the status of data in a specific coupling facility structure, not units of work specific to a given CQS.

Before CQS can restart, CQS must recover each queue structure defined to CQS, if needed. CQS connects to both the primary and the overflow queue structures for each structure pair defined. CQS determines whether the structures need to be warm or cold started and performs the necessary recovery functions. If more than one structure pair is defined to CQS, one structure pair can be warm started and another can be cold started.

CQS Structure Allocation

A structure is allocated the first time a CQS connects to it. The structure is persistent. It remains allocated until you explicitly delete it using the MVS SETXCF command.

When a CQS connects to a structure, the structure might be empty, that is, it might contain no data, might contain only CQS control information, or it might contain client data. The structure can be empty if:

- This is the first time a CQS is accessing the structure.
- You scratched the structure to perform a structure cold start.
- A structure failure occurred and the structure must be recovered.

CQS Structure Warm Start

CQS warm starts both a primary structure and its overflow structure, if the primary structure contains data or if one SRDS contains valid structure checkpoint data and the CQS log contains valid data.
During a structure warm start, CQS determines the status of the structure and
initiates a structure recovery if necessary. If a structure recovery is needed, CQS
allocates the structure and repopulates it from either the SRDS, that contains valid
client data from a previous checkpoint, and the CQS log or from the CQS log by
itself.

After a structure warm start has completed, CQS determines whether a future
recovery is possible based on the status of the structure recovery data sets and the
log stream. If the primary structure contains client data, but neither the SRDS nor
the log can be used for future recovery, CQS issues a CQS0009W message.

**Recommendation:** If CQS issues a CQS0009W message, initiate a structure
checkpoint as soon as possible. If a structure checkpoint does not complete
successfully and the structure fails, CQS cannot recover the structure.

After a structure warm start, CQS can be cold started or warm started. If the log
records needed for the CQS restart have been deleted, you might have to cold start
the CQS.

### CQS Structure Cold Start

CQS cold starts both a primary structure and its overflow structure, if the primary
structure is empty and both of the structure recovery data sets are empty. During a
structure cold start, CQS deletes:

- The overflow structure
- All the log records in the log stream for the structure

To cold start a structure, you must:

1. Ensure that all CQSs are disconnected from the structure.
2. Delete the primary and overflow structures on the coupling facility.
3. Scratch both structure recovery data sets (SRDS 1 and 2) for the structure.

During a structure cold start, CQS deletes:

- The overflow structure
- All log records in the log stream for the structure

When structure cold start completes, CQS automatically performs cold start restart
processing.

### CQS Structure Recovery for Restarting

A structure might need to be recovered if the structure is empty or if it contains only
CQS control information. Data from the last structure checkpoint and the MVS log
stream are used to recover a structure. First the structure is repopulated with data
objects from the structure recovery data sets. CQS then reads the log, starting at
the time of the structure checkpoint, and updates the structure with changes that
occurred after the structure checkpoint.

If the primary structure is empty and neither SRDS contains valid structure
checkpoint data, CQS determines whether it can use just the CQS log for recovery.
If the first log record in the log stream is the **Beginning of Log** record, the log
stream contains all the log records required for recovery and CQS can use the log
records to complete the structure recovery.
If CQS finds that a previous structure rebuild did not complete successfully, it initiates another rebuild.

If the primary structure contains only CQS control information and the first CQS that connected to the structure (the one that allocated the structure) is not able to determine whether a rebuild is needed, CQS initiates a rebuild if either SRDS is valid or if all the log records are available.

If neither SRDS is valid and the log records have been deleted by a previous structure checkpoint, CQS cannot rebuild the structure. When this happens and a rebuild is necessary, CQS issues WTOR CQS0034A message asking what you want to do. You can cold start the structure or cancel this CQS. For more information on structure recovery, see “CQS Structure Recovery” on page 45.

**Restarting CQS**

After CQS completes the structure initialization, it continues with the restart. CQS can either do a cold restart or a warm restart.

Restarting CQS affects only the units of work that this CQS manages. Restarting does not back out or restore any units of work owned by another CQS. When you have completed restarting all structure pairs for a particular CQS, the CQS ready message is issued (CQS0020I).

Since changes to resource structures are not checkpointed or logged, restarting CQS does not affect units of work for resource structures.

The frequency of system checkpoints affects restart. CQS must read more log records when checkpoints are infrequent than when checkpoints occur more often. Also, the amount of logging that one CQS performs can affect another CQS during restart. All CQSs write to the same log, so a CQS restarting must read all log records written by all CQSs.

CQS takes an initial system checkpoint at the end of a restart.

**CQS Warm Start**

During a warm start, CQS reads the log records from the last system checkpoint, restores the environment for committed data objects, and backs out uncommitted data objects. Doing so prepares CQS to regain synchronization with the client and resume processing. Normally, CQS warm start is automatic and you do not need to take any action.

When CQS warm starts, it reads the checkpoint data set to find the log token representing the last system checkpoint. When CQS finds this log token, it initiates a warm start. If CQS fails to find this log token in the checkpoint data set, it reads the log token from the structure. If CQS finds the log token, CQS issues WTOR (Write to operator with reply) CQS0031A to allow you to confirm the use of this token. At this point, you can do one of the following:

- Confirm the log token.
- Cold start CQS.
- Cancel CQS.
- Specify a new log token.
If you specify a new log token and CQS fails to find this log token, CQS issues WTOR CQS0032A. At this point, you can do one of the following:

- Cold start CQS.
- Cancel CQS.
- Specify a new log token.

Sometimes CQS purges log records that are required for restart. CQS purges log records in the following situations:

- During a structure checkpoint
- When the log becomes full and no more data sets are available for logging

**Related Reading:** See “Using CQS Structure Checkpoint” on page 39 for more details on structure checkpoint and the purging of log records.

**Important:** CQS might not have any log records if it is only managing resource structures.

**Recommendation:** If a CQS does not accept a log token during CQS restart, cold start the CQS. In cases where multiple CQSs are running, it is possible that log records for a CQS that previously failed and was not restarted are purged while another CQS performs a structure checkpoint.

### CQS Cold Start

When CQS restarts after a structure cold start, CQS cold start processing is automatic. You do not need to take any action. CQS takes a system checkpoint and then CQS restart is complete.

When CQS cold starts after a structure warm start or a structure recovery, CQS reads the structure to find unresolved work. CQS backs out requests to move data but completes requests to delete data. CQS performs a system checkpoint and restart is complete.

No log records are read or processed when CQS is cold started.

To cold start CQS, you must:

1. Scratch the CQS system checkpoint data set for the structure.
2. Reply “COLD” to the CQS0031A WTOR.

### Using the MVS Automatic Restart Manager with CQS

CQS, if requested, can register with the MVS Automatic Restart Manager. The Automatic Restart Manager (ARM) is an MVS recovery function that can improve the availability of started tasks. When a task fails or the system on which it is running fails, the Automatic Restart Manager can restart the task without operator intervention.

**Recommendation:** Register with the MVS Automatic Restart Manager regardless of the types of structures CQS is using.

To enable the Automatic Restart Manager, you can specify ARMST=Y in one of two ways:

- In the CQSIxxx PROCLIB member
- As an execution parameter
An abend table exists in module CQSARM10. The table lists the CQS abends for which the Automatic Restart Manager does not restart CQS after the abend occurs. You can modify this table.

IBM provides policy defaults for automatic restart management. You can use these defaults or define an Automatic Restart Manager policy to specify how CQS should be restarted. The policy can specify different actions to be taken when the system fails and when CQS fails. When Automatic Restart Manager is enabled, CQS registers to ARM with an ARM element name of CQS + cqssn + CQS. Use this ARM element name in the ARM policy to define the automatic restart manager policy for CQS.

cqssn is the CQS name defined as a CQS execute parameter or in the CQSIPxxx PROCLIB member with the SSN= parameter. For example, if SSN=CQS1A, then the ARM element name is CQSCQ1ACQS.

**Related Reading:** For more information on using the MVS Automatic Restart Manager with CQS, see "CQS Execution Parameters" on page 17. See MVS Setting Up a Sysplex, GC28-1779 for OS/390 or SA22-7625 for z/OS, for details MVS Automatic Restart Manager policy.

**Restarting CQS After CQS Resource Cleanup Failures**

If CQS abends and you receive message CQS0102E with module = CQSRSM00, log records might be missing from the CQS log. Message CQS0102E indicates that a failure occurred during CQS resource cleanup. This failure might prevent CQS log records in internal buffers from being externalized to the CQS log.

If this situation occurs, perform one of the following actions:

- If the terminating CQS is the only CQS running for its set of structures, restart the CQS immediately.
- If other CQSs are running, either immediately restart the terminated CQS, or initiate a structure checkpoint on one of the surviving CQSs. For more information, see the /CQCHKPT SHAREDQ command in IMS Version 8: Command Reference.

Successfully restarting the failed CQS or taking a structure checkpoint is necessary to preserve the state of the data on the shared queues in the event that a structure rebuild is needed.

**Establishing Client Connection to CQS During Failed Client Takeover**

When the client (such as IMS) supports XRF takeover capability, one client must take over the work for a failed client. The CQS connected to this failed client might not be active. Therefore, during the takeover process, the client taking over work from a failed client must connect to a different CQS and indicate that it is taking over work from another client. At this point, CQS must perform a process similar to CQS restart, using the log records from the CQS connected to the failed client. Normally, CQS failed client connection restart (warm start) is automatic and you do not need to take any action.

During a failed client connection, CQS reads the log token from the structure for the CQS connected to the failed client.

- If CQS finds the log token, CQS performs warm start processing for the failed client.
If CQS does not find the log token, CQS issues WTOR CQS0033A. At this point, you can do one of the following:

- Cold start the client connection.
- Reject the client connection request.
- Specify a new log token.

**Recommendation:** If a CQS does not accept a log token during failed client connection restart, cold start the connection. If multiple CQSs are running, one CQS structure checkpoint might purge the log records for another CQS that previously failed and was not restarted.

### Authorizing Access To CQS

If RACF® or another security product is installed at your installation, the security administrator can define profiles that control the ability of clients to connect to and access CQS structures.

### Authorizing CQS Registration

When a client issues the CQSREG request to register with CQS, CQS issues a RACROUTE REQUEST=AUTH call to determine whether the client is authorized to register with CQS. RACF checks the user ID of the client that issued the CQSREG request. This user ID must have at least UPDATE authority to register with CQS.

The RACF security administrator can define profiles in the FACILITY class to control registration with CQS. The profile names must be of the form CQS_.cqs_id, where cqs_id is the ID of the CQS that is to be protected. The cqs_id value is the subsystem name (SSN) as defined in the CQSIPxx PROCLIB member, followed by the characters CQS. For example, if the SSN is ABC, the cqs_id value is ABCCQS.

**Example:** To define a profile for CQS to prevent users other than CQSUSER1 and CQSUSER2 from registering, issue the RACF commands shown in Figure 12:

```
RDEFINE FACILITY CQS.ABCCQS UACC(NONE)
PERMIT CQS.ABCCQS CLASS(FACILITY) ID(CQSUSER1) ACCESS(UPDATE)
PERMIT CQS.ABCCQS CLASS(FACILITY) ID(CQSUSER2) ACCESS(UPDATE)
SETROPTS CLASSACT(FACILITY)
```

**Figure 12. RACF Commands for Authorizing CQS Registration**

### Authorizing Connections to CQS Structures

When a client issues the CQSCONN request to connect to a CQS structure, CQS issues a RACROUTE REQUEST=AUTH call to determine whether the client is authorized to access the structure. RACF checks the user ID of the client that issued the CQSCONN request. This user ID must have at least UPDATE authority to connect to the structure through CQS.

The RACF security administrator should define profiles in the FACILITY class to control the connection to CQS structures. The profile names must be of the form CQSSTR.structure_name, where structure_name is the name of the primary CQS structure that is to be protected. Use the same structure name that you define in the CQSSGxxx and CQSSLxxx PROCLIB members.

The CQSSTR.structure_name profiles only control access to the specified structures through CQS; they do not control direct access to the structures using IXL macros.
You can provide control over direct structure access by defining RACF profiles of the form IXLSTR.structure_name. If you create such profiles, you must give the user IDs under which you run CQS access to the structures.

**Related Reading:** For information on protecting direct access to coupling facility structures, see “Authorizing Coupling Facility Requests” in the *MVS Programming: Sysplex Services Guide.*

For more information on defining structure names, see “CQS Global Structure Definition PROCLIB Member (CQSSGxxx)” on page 22 and “CQS Local Structure Definition PROCLIB Member (CQSSLxxx)” on page 20. CQS does not perform a separate check on the overflow structure name, because the primary and overflow structures are considered one unit.

**Example:** To define a profile for a CQS primary structure named IMSMSGQ01, and to allow only user CQSUSER to connect to it, issue the RACF commands shown in Figure 13:

```
RDEFINE FACILITY CQSSTR.IMSMSGQ01 UACC(NONE)
PERMIT CQSSTR.IMSMSGQ01 CLASS(FACILITY) ID(CQSUSER) ACCESS(UPDATE)
SETROPTS CLASSACT(FACILITY)
```

*Figure 13. RACF Commands to Authorize Connection to CQS Structures*

If you do not define a profile for a particular CQS structure, the structure is not protected, and any user ID can issue a CQSCONN request to access the structure.

### Using Structure Alter for CQS

Structure alter is an OS/390 process supported by CQS that can be used to alter the structure size or to redistribute the objects within the structure. CQS supports structure alter for primary queue structures, overflow queue structures, and resource structures. CQS allows you to dynamically change the size of a primary or overflow structure.

To enable structure alter, activate a CFRM policy and define the INITSIZE and SIZE parameters in this policy. For information on structure size, see “Determining Structure Size for CQS Connections” on page 15.

To initiate the structure size change, enter the following XES command:

```
SETXCF START,ALTER,STRNAME=strname,SIZE=size
```

The value of size must be within the range of values between INITSIZE and SIZE in the CFRM policy.

Automatic structure alter is an OS/390 function that can automatically alter the structure size or the element to entry ratio when the structure full threshold is reached. CQS supports automatic structure alter for queue structures and resource structures. To enable automatic structure alter, activate a CFRM policy defined with INITSIZE, SIZE, ALLOWAUTOALT(YES).

**Important:** A structure enabled with automatic structure alter is a candidate to be contracted in size by OS/390, if the coupling facility storage becomes constrained. Be careful when enabling automatic structure alter for queue structures. If OS/390 contracts the queue structure size, it might cause the queue structure to go into
overflow unnecessarily. To prevent this happening, define the CFRM policy with a MINSIZE (minimum size), below which OS/390 will not contract the structure.

### Using CQS System Checkpoint

This section introduces CQS system checkpoint, the checkpoint data sets that are used for recovery, and how CQS restarts after system checkpoint.

System checkpoint applies to a CQS if it manages at least one queue structure. If a CQS manages only a resource structure, system checkpoint does not apply.

At a system checkpoint for recovering CQS information, CQS writes log records that contain restart and recovery information to the CQS log. CQS does not stop activity while the checkpoint is in progress.

CQS performs a system checkpoint in each of the following situations:

- When a client issues a CQSCHECKPT FUNC=CHKPTSYS request
- When the number of log records that CQS writes reaches the value specified on the SYSCHKPT= parameter in the CQSSLxxx PROCLIB member
- When the client is IMS and you enter the /CQCHKPT SYSTEM command
- When a client RESYNC ends
- When structure checkpoint ends successfully
- At the end of a restart

In addition, CQS takes system checkpoints during significant events, such as a shutdown.

### CQS Checkpoint Data Set

For each structure pair, CQS maintains a checkpoint data set. CQS writes to its checkpoint data set and uses it during restart.

The checkpoint data set is dynamically allocated during CQS initialization. You define the checkpoint data set DSNNAME for a structure using the CHKPTDSN= parameter in PROCLIB member CQSSLxxx.

### How CQS Restarts after System Checkpoint

During CQS restart, CQS reads the log records from the last system checkpoint and restores the environment for committed data objects and backs out uncommitted data objects on queue structures. The frequency of system checkpoint affects this restart. CQS must read more log records when checkpoints are infrequent than when the checkpoints occur more often. Because the CQS log is shared by multiple CQSs, CQS restart time is affected by the number of log records written by the multiple CQSs, not just the CQS that is being restarted.

CQS takes an initial system checkpoint at the end of a restart.

### Using CQS Structure Checkpoint

Structure checkpoint takes a snapshot of the shared queues on a queue structure and writes the data to the structure recovery data set (SRDS) so that CQS can recover the queues after a structure failure. Structure checkpoint processing copies all recoverable data objects from a structure pair to a structure recovery data set (SRDS). For nonrecoverable data objects, the queue name, and UOW are copied,
Recovering CQS and Structures

but not the actual data object. The client specifies whether or not a data object is recoverable when the CQSPUT FUNC=PUT request is issued to insert the data object onto the shared queues. For example, when IMS is the client, all data objects are marked as recoverable, except for Fast Path input messages.

**Important:** Structure checkpoint is not supported for resource structures. It supports queue structures only.

When it performs the copy operation, CQS stops all activity against the structure to ensure that the structure does not change while the checkpoint is being taken. If CQS receives a request to process work when a structure checkpoint is in progress, the request is held until after the structure checkpoint is complete.

**Recommendation:** Because no other work for a structure can be processed while CQS is taking a checkpoint, consider processing structure checkpoints during non-peak hours.

After all shared queues are copied to the SRDS, each CQS performs a system checkpoint to ensure its restart checkpoint has a time stamp that is more recent than the current structure checkpoint. The structure checkpoint process then deletes all log records that are not needed for structure recovery, allowing the logger to reclaim space in the CQS log and preventing the log from becoming full. After log records are deleted, CQS cannot access these log records and, therefore, cannot use these records for structure recovery or CQS restart. If only one SRDS contains valid structure checkpoint data, all log records that were written prior to that structure checkpoint are deleted. If both SRDSs contain valid structure checkpoint data, all log records that were written prior to the oldest structure checkpoint are deleted.

If a CQS was not active at the time of a structure checkpoint, it cannot initiate a system checkpoint, meaning that its restart checkpoint is older than at least one structure checkpoint. If both SRDSs contain valid structure checkpoint data, no problem exists (because the CQS restart checkpoint is still more recent than the oldest structure checkpoint, so its restart log records are not deleted). However, if this is the first or only valid structure checkpoint, or a CQS was down across two structure checkpoints, the log records needed for that CQS to restart are deleted. In this case, that CQS might need a cold start to restart.

**Recommendation:** Initiate a structure checkpoint after a structure cold start, or anytime the SRDSs are deleted and redefined. The structure checkpoint should start after all CQSs that share the structure are started. This provides the SRDS with an initial structure checkpoint. Also, to update the snapshot of the shared queues and to periodically delete log records, initiate a structure checkpoint at regular intervals.

When a structure recovery is required, the SRDS and the CQS log are used to recover the shared queues. CQS first repopulates the new structure from the SRDS. CQS then reads all log records from the time the structure checkpoint completed. The length of time to read the log records is dependent on how many log records are in the log. More frequent structure checkpoints reduce the number of log records that must be read during a structure recovery. Deleting the log records also helps prevent the log from becoming full. When a log stream becomes full, CQS deletes all log records older than the oldest structure checkpoint or CQS system checkpoint. CQS then takes a structure checkpoint.

CQS performs structure checkpoints in each of the following situations:
Recovering CQS and Structures

- When the MVS log becomes full or approaches full.
- After a successful structure recovery.
- After a successful overflow threshold process.
- When a client issues the CQSCHECKPT FUNC=CHKPTSTR request.
- When the client is IMS, and you enter the /CQCHKPPT SHAREDQ command.
- During CQS normal termination when the client requests it on the CQS DISC request. When the client is IMS, you can request a structure checkpoint at CQS termination by entering the /CQSET SHUTDOWN SHAREDQ command. IMS then passes this request to CQS when IMS terminates normally with a /CHECKPOINT FREEZE|DUMPQ|PURGE command.

### Preventing CQS Structure Full

You should manage structure usage to avoid a structure full condition. If a resource structure becomes full, CQS issues message CQS0205E. CQS provides two ways for you to prevent a structure full condition:

- CQS structure overflow function for queue structures
- OS/390 structure full monitoring capability, used with CQS, for queue structures and resource structures

Use these mechanisms to warn when a structure full condition is approaching and to take action to prevent a full structure.

### CQS Structure Overflow Function

CQS provides a structure overflow function that automatically warns you when a queue structure is approaching full and takes action to prevent a full structure.

When the usage of a structure reaches the overflow threshold, CQS attempts to make the structure larger by initiating a structure alter. If the alter fails, CQS either allocates an overflow structure and moves selected queues to the overflow structure (if you define an overflow structure) or rejects data objects from being put on the selected queues. CQS stops all activity against the structure during this processing.

**Definition:** The **overflow threshold** is the percentage of the primary structure that must be in use before CQS goes into overflow mode. The default overflow threshold is 70%, but you can change the default by defining the OVFLWMAX parameter in the CQSSGxxx PROCLIB member.

**Important:** Structure overflow is not supported for resource structures.

If CQS does not succeed in altering a structure’s size, the structure goes into overflow mode. In overflow mode, CQS selects queues using the most space on the structure as candidates for overflow processing. CQS stops selecting queues when enough queues have been selected to cause the primary structure usage to fall 20% below the overflow threshold. Activity against the structure is temporarily stopped while queues are being selected for overflow. CQS drives the Queue Overflow User-Supplied exit routine with the candidate queue names, which the exit then approves or rejects for overflow processing. Queues that get approved are placed into overflow mode. If an overflow structure is defined, CQS allocates the overflow structure and moves the approved queues to the overflow structure. If an overflow structure is not defined, CQS rejects CQSPUT requests for the approved queues. Overflow structures can be defined in the CQSSGxxx PROCLIB member, using the OVFLWSTR parameter.
CQS exits overflow mode either after all of the queues have been removed from the overflow structure (if an overflow structure gets allocated), or when the primary structure usage has gone 20% below the overflow threshold (if there is no overflow structure).

**CQS Structure Full Monitoring**

The OS/390 structure full monitoring capability can be used for queue structures and resource structures to warn you when a structure is approaching full and to prevent a full structure.

If structure full monitoring is enabled, OS/390 monitors structure usage. When the number of entries or elements in use reaches the structure full threshold, OS/390 issues a highlighted IXC585E message to warn the system programmer that a structure full condition is imminent. If automatic altering is enabled, OS/390 automatically initiates a structure alter to increase the structure size or change the element to entry ratio, and also enable structure full monitoring. The structure full monitoring threshold is defined with the CFRM policy FULLTHRESHOLD keyword. Structure full monitoring is automatically enabled with a default threshold of 80%. Define a different threshold with the CFRM policy FULLTHRESHOLD parameter. Define the CFRM policy with THRESHRESHOLD(0) to disable structure full monitoring. When the structure usage goes below the threshold, OS/390 issues an IXC586I message.

The following command displays the structure full threshold that is in effect:

```
D XCF,STRUCTURE,STRNAME=strname
```

**Examples** In the example display shown in Figure 14, the command D XCF,STRUCTURE,STRNAME=IMSRSRC01 is issued and the structure full threshold is 80%.

```
STRNAME: IMSRSRC01
STATUS: NOT ALLOCATED
POLICY SIZE : 4096 K
POLICY INITSIZE : N/A
FULLTHRESHOLD : 80
REBUILD PERCENT : N/A
DUPLEX : DISABLED
PREFERENCE LIST : LF03
ENFORCEORDER : NO
EXCLUSION LIST IS EMPTY
```

*Figure 14. Display for Structure Full Threshold - Example 1*

```
*IXC585E STRUCTURE IMSRSRC01 IN COUPLING FACILITY LF03, 725
PHYSICAL STRUCTURE VERSION B4704775 92D95302,
IS AT OR ABOVE STRUCTURE FULL MONITORING THRESHOLD OF 80%.
ENTRIES: IN USE: 4874 TOTAL: 4874, 100% FULL
ELEMENTS: IN USE: 19 TOTAL: 4872, 0% FULL
```

*Figure 15. Display for Structure Full Threshold - Example 2*

```
Figure 16 on page 43 shows the IXC586I message:
```
Using Structure Full Monitoring with CQS Structure Overflow

You can use the structure full monitoring function (an OS/390 function) with the structure overflow function (a CQS function) for queue structures.

The overflow threshold is a value defined to CQS. The structure full threshold is a value defined to OS/390. If the overflow threshold is close to the structure full threshold and automatic altering is enabled, CQS and OS/390 may both try to initiate a structure alter at the same time to prevent the structure from becoming full.

If a CQS-initiated structure alter is in progress when OS/390 detects the structure full threshold has been reached, OS/390 stops the CQS-initiated structure alter and initiates its own structure alter. When CQS detects that its structure alter has failed, CQS goes into overflow mode, even if the OS/390-initiated structure alter reduces the structure usage below the overflow threshold.

**Recommendation:** Consider your structure full threshold when deciding what overflow threshold to define, so that you control when a structure goes into overflow mode. If you use structure full threshold, define it to be lower than the overflow threshold to avoid going into overflow mode unnecessarily. If the structure full threshold is lower than the overflow threshold, OS/390 can attempt structure full threshold processing before the structure goes into overflow mode.

**Related Reading:**
- For detailed information about the CQSSGxxx PROCLIB member, see "CQS Global Structure Definition PROCLIB Member (CQSSGxxx)" on page 22.
- For detailed information about the Queue Overflow User-Supplied exit routine, see "Queue Overflow User-Supplied Exit Routine for CQS" on page 55.
- For detailed information about the CQSPUT request, see "CQSPUT Request" on page 120.

Rebuilding Structures in CQS

Structure rebuild is an OS/390 process that allows another instance of a structure to be allocated with the same name and data reconstructed from the initial structure instance. OS/390 supports system-managed rebuild, CQS-managed rebuild, and structure duplexing. CQS supports system-managed rebuild and CQS-managed rebuild for queue structures and resource structures. Note that CQS stops all activity against the structure during structure rebuild.

**OS/390 System-Managed Rebuild and CQS**

System-managed rebuild is an OS/390 process by which OS/390 rebuilds the structure. OS/390 copies the structure contents to a new structure.

System-managed rebuild is supported for queue structures and resource structures.
System-managed rebuild is only done if no CQS is up. If a CQS is up, the CQS performs a user-managed rebuild and does the structure copy.

Use system-managed rebuild primarily for planned reconfiguration. If the rebuild is initiated with the `SETXCF` START,REBUILD command and no CQS is available to perform the structure copy, OS/390 performs the structure copy.

**Restrictions**: System-managed rebuild does not address coupling facility failures, structure failures, or loss of connectivity. CQS-managed rebuild is required to handle such failures.

To enable a structure for system-managed rebuild, add the following parameter to your CFRM couple data set utility job, then run the JCL to format the CFRM couple data set with system-managed rebuild capability.

```
ITEM NAME(SMREBLD) NUMBER(1)
```

**CQS-Managed Rebuild**

CQS-managed rebuild is a process by which CQS manages structure rebuild. CQS supports two variations of CQS-managed rebuild: structure copy and structure recovery. Structure copy copies the contents of the structure to another structure, for a planned reconfiguration or connectivity loss. Structure copy can also be used to activate new CFRM policy attributes. Structure recovery recovers a structure from the SRDS and the MVS log after a structure failure.

If one CQS loses connectivity to a structure and another CQS still has connectivity to that structure, CQS manages the structure rebuild and performs a structure copy. If all CQSs lose connectivity to a resource structure, structure recovery is attempted, but fails because structure recovery is not supported for resource structures.

If a coupling facility or queue structure fails, CQS performs a structure recovery.

If a resource structure fails, it is lost and structure rebuild is not performed. CQS is not able to perform structure recovery because resource structures do not support checkpoint and logging. CQS clients can repopulate the failed resource structure. CQS attempts to allocate a new resource structure.

If a new structure is successfully allocated, CQS drives the client structure event exit with the repopulate structure event. The CQS client or clients must then repopulate the structure. If a new structure is not successfully allocated, CQS drives the structure exit event with the structure failed event. The structure is not accessible for repopulation. Correct the environmental problem that caused the structure allocate to fail so that the structure can be allocated and repopulated.

**Initiating Structure Rebuild with OS/390 and CQS**

A structure rebuild can be initiated by an OS/390 operator, by CQS, or by OS/390:

- An OS/390 operator can initiate a structure rebuild to copy or recover queues using the following command:
  
  `SETXCF START,REBUILD,STRNAME=stname,LOCATION=NORMAL/OTHER`

- CQS initiates a structure rebuild if, during CQS initialization, it detects an empty structure and a valid SRDS (indicating a valid structure checkpoint in the SRDS). If CQS detects an empty structure and a valid SRDS, it also initiates a structure rebuild during ENF 35 event processing.
OS/390 initiates a structure rebuild if the rebuild threshold for loss of connectivity is reached. The rebuild threshold for loss of connectivity is defined with the CFRM policy REBUILDPERCENT keyword. The REBUILDPERCENT default is 1. If the system programmer does not define REBUILDPERCENT, OS/390 initiates a rebuild if any CQS loses connectivity to the structure.

- If structure copy aborts because of a CQS failure and no other CQS can determine if the failed CQS is the master, then the rebuild starts over as a structure recovery.

### CQS Structure Repopulation

Structure repopulation is a process by which CQS clients repopulate a failed resource structure. CQS does not support structure recovery for resource structures because CQS does not log or checkpoint resource updates.

If a resource structure and its duplex fail, the CQS clients can repopulate the resource structure. CQS attempts to allocate a new structure. If this allocation is successful, CQS notifies its clients to repopulate. The CQS client or clients must then repopulate the structure. Any resources that were kept only on the resource structure are lost.

If CQS fails to allocate a new structure, CQS notifies the client that the structure failed. If the sysplex environment changes later and CQS is eventually able to allocate a new resource structure, CQS notifies the client to repopulate at that time. Alternately, correct the environmental problem that caused the structure allocate to fail so that the structure can be allocated and repopulated.

CQS does not coordinate resource structure repopulation between CQS clients; clients must synchronize resource structure repopulation if desired. Structure repopulation does not guarantee the restoration of all objects; some objects may be lost.

### CQS Structure Recovery

The structure recovery function recovers the data objects on a structure from the SRDS and the MVS logs after a structure failure.

**Important:** Structure recovery is not supported for resource structures.

After a structure failure, the structure might need to be recovered if it is empty or contains only CQS control information. During structure recovery, CQS allocates a structure and repopulates it from either the SRDS (containing valid client data from a previous checkpoint) and the CQS log or the CQS log by itself.

When CQS recovers the structure from a structure checkpoint, it repopulates the structure with the data objects from the structure recovery data set. CQS reads the log starting at the time of the structure checkpoint to update the structure with changes that occurred after the structure checkpoint.

If the primary structure is empty and neither SRDS contains valid structure checkpoint data, CQS determines whether it can use just the CQS log for recovery. If the first log record in the log stream is the Beginning of Log log record, the log stream contains all of the log records required for recovery and CQS can use the log record to complete the structure recovery.

If CQS finds that a previous structure rebuild did not complete successfully, it initiates another rebuild.
Rebuilding Structures

If the primary structure contains only CQS control information and the CQS that allocated the structure is not able to determine if a rebuild is necessary, CQS initiates a rebuild if either SRDS is valid or all log records are available.

If neither SRDS is valid and the log records are deleted by a previous structure checkpoint, CQS cannot rebuild the structure. In this case, if rebuild is necessary, CQS issues WTOR CQS0034A to ask you what to do. You can cold start the structure or cancel this CQS.

If no CQS has access to the structure when structure rebuild is initiated, the structure is recovered from the SRDS and the CQS log. Nonrecoverable data objects (such as IMS Fast Path input messages) are lost. Data objects are read from the SRDS and copied into a new structure. CQS then reads the log to bring the structure back to the point of currency. The log contains all the records necessary for structure recovery if no structure checkpoint was ever initiated. In this case, the structure is recovered from just the CQS log.

A client can use the CQSCONN request to specify whether work can be performed while a structure is being rebuilt. While structure recovery is in progress, CQS stops all activity against the structure. This means that CQS requests are held until the structure recovery is complete. You can allow CQS requests to continue during structure rebuild by specifying WAITRBLD=N0 when connecting to the structure with the CQSCONN request. In this case, structure recovery stops structure activity for some time, but the structure becomes available much sooner.

CQS Structure Copy

The structure copy function copies all of the data objects (both recoverable and nonrecoverable) from the structure to a new structure for a planned reconfiguration or unplanned activity such as loss of connectivity. Structure copy can be used to change the location of the structure or any other attribute defined in the CFRM policy, such as SIZE, INITSIZE, and PREFLIST. When a structure rebuild is initiated, at least one CQS must have access to the structure for structure copy to be performed.

OS/390 Structure Duplexing for CQS

Structure duplexing is an optional OS/390-managed process for failure recovery of shared queue structures and resource structures. In this process, OS/390 creates a duplex copy of a structure in advance of a failure, then maintains the structures in a duplexed state during normal operation.

If a queue structure fails and duplexing is enabled, OS/390 switches to the unaffected structure instance. If a queue structure fails and duplexing is not enabled, CQS rebuilds the structure based on data from the most recent checkpoint and OS/390 log entries. The advantage of duplexing queue structures in the event of a failure is in avoiding the overhead of a CQS-managed structure rebuild.

If duplexing is enabled and a resource structure fails, OS/390 switches to the unaffected structure instance. If duplexing is not enabled and a resource structure fails, the data objects are lost because resource structures do not support checkpoint or logging. CQS repopulates the resource structure with control information. CQS notifies its clients to repopulate the structure. It is up to the clients to repopulate the resource structure if necessary.

Recommendation: Enable structure duplexing for resource structures.
If both instances of a structure fail at the same time, structure duplexing does not work and all data objects are lost. If the failed structure is a resource structure, the CQS client must repopulate it. If the failed structure is a queue structure, CQS recovers the structure using structure rebuild.

Structure duplexing is optional. To use it, you must enable the OS/390 1.2 duplexing function. Perform the following steps to enable this function:

1. Ensure that the sysplex is defined as duplexing capable.
2. Add the following parameter to your CFRM couple data set format utility:
   ITEM NAME(SMDUPLEX) NUMBER(1)
3. Migrate to an environment in which system-managed duplexing is enabled from a CFRM standpoint. A nondisruptive migration of CFRM couple data sets is required. Only OS/390 systems at a level that supports system-managed duplexing are capable of using system-managed CFRM couple data sets that are duplexing-capable. Therefore, take the following steps:
   a. Incrementally migrate all systems in the sysplex that are using CFRM to the OS/390 level that supports system-managed duplexing.
   b. Format system-managed duplexing-capable CFRM couple data sets and bring them into use as the primary and alternate CFRM couple data sets for the configuration.

Important: After steps 1 through 3 are completed, you cannot return to down-level CFRM couple data sets (ones that are not system-managed duplexing-capable) without disruption. Doing so requires a sysplex-wide IPL of all systems using the system-managed duplexing-capable data sets.

Once an up-level CFRM couple data set is in use in the sysplex, system-managed duplexing can be started and stopped in a nondisruptive manner. To turn this function on or off, even while the CFRM couple data set is in use, modify the CFRM policy DUPLEX parameter or use the SETXCF START,STOP,REBUILD,DUPLEX operator command.

To enable system-managed duplexing for a particular structure, the structure must be defined as duplexing-capable. Defining a structure as duplexing capable also defines it as system-managed rebuild-capable. Add the following parameter to your CFRM active policy:

DUPLEX (ENABLED)

or

DUPLEX(ALLOWED)

If DUPLEX(ENABLED) is defined in the CFRM active policy, the system programmer or OS/390 internally can initiate the duplexing rebuild. OS/390 triggers the start of duplexing rebuild based on a timer or upon detection of certain events (such as connect, disconnect, and policy change). When CQS initializes and connects to a structure defined with DUPLEX(ENABLED), OS/390 starts a duplexing rebuild.

If DUPLEX(ALLOWED) is defined in the CFRM active policy, the duplexing rebuild must be initialized by the system programmer using the following command:

SETXCF START,REBUILD,DUPLEX,STRNAME=strname

Important: If you define overflow structures with DUPLEX(ENABLED), IMS initialization allocates the overflow structure and duplexing begins. If IMS initialization determines that the overflow structure is not needed, it deletes it and
Rebuilding Structures

duplexing terminates. If you want to avoid this unnecessary overhead, during CQS initialization define the overflow structure with DUPLEX(ALLOWED) and initiate duplexing with a SETXCF command when the structure goes into overflow mode.

After duplexing is established, the structure remains in that state indefinitely. Duplexing can be stopped internally by OS/390 if an error occurs (such as link failure, structure failure, and CFRM policy change). The system programmer can explicitly stop duplexing using the following command:

```
SETXCF STOP,REBUILD,DUPLEX,STRNAME=strname,KEEP=OLD|NEW
```

where you specify KEEP=OLD to keep the old structure and KEEP=NEW to keep the new structure.

Planned reconfiguration (such as a CFRM policy change or taking a coupling facility offline for maintenance) is supported. Structure rebuild is not permitted for a structure that has established duplexing, so the duplexing must be stopped first. Perform the following steps:

1. Stop duplexing.
   Stop duplexing and switch the structure to simplex mode by issuing the following command:
   ```
   SETXCF STOP,REBUILD,DUPLEX,STRNAME=strname,KEEP=OLD|NEW
   ```

2. Reconfigure.
   Make the change required for planned reconfiguration.

3. Initiate duplexing rebuild.
   Initiate a new duplexing rebuild by issuing the following command:
   ```
   SETXCF START,REBUILD,DUPLEX,STRNAME=strname
   ```

Deleting a Structure When CQS Is Not Connected

You can delete a structure when no CQS is connected to it. To delete a structure:

1. Shut down all CQSs connected to the structure.

2. If there are any failed persistent connections, then they must be deleted before the structure can be deleted. Enter the SETXCF FORCE,CONNECTION,STRNAME=strname,CONNAME=ALL command.

   **Attention:** When a CQS fails while connected to a structure, it should be allowed to restart so it can clean up any work that was in process at the time it failed. This command can be used to terminate the failed connections when you have to delete the structure.

   If this command is used incorrectly, the queues or resources may be lost.

3. Enter the SETXCF FORCE,STRUCTURE,STRNAME=strname command.
   Ensure that the strname in this command is the same as the strname specified in the CQS global structure definition PROCLIB member and the CQS local structure definition PROCLIB member.

Shutting Down CQS

A CQS client can use the CQSSHUT request, the CQSDISC with the CQSSHUT=YES parameter to shut down CQS, or you can issue the MVS STOP command to shut down CQS.

**Related Reading:** See [“Shutting Down CQS” on page 82](#) for information on how the client can shut down CQS.
Normally, when a client disconnects from CQS using the CQSDISC request and specifying CQSSHUT=YES, CQS shuts down after no clients are connected to it.

In some cases, however, the CQS address space remains active, even when no clients are connected to it. This can happen under any of the following conditions:

- No client is connected to CQS when CQS is started.
- A client that had been connected to CQS terminates abnormally, without issuing a CQSDISC request to disconnect from CQS, or issues a CQSDISC request with CQSSHUT=NO specified.

You can shut down a CQS address space that has no clients connected to it by issuing the MVS STOP command, specifying the job name of the CQS address space.

**Example:** P cqsjobname

cqsjobname is the job name of the CQS address space you want to stop. If no clients are connected to a CQS, that CQS shuts down. If clients are connected to the CQS, the stop command is rejected, and message CQS0300I is issued.
Chapter 4. CQS User-Supplied Exit Routines

Note: Throughout this chapter the term “user exit routine” means “user-supplied exit routine.”

This chapter describes the Common Queue Server (CQS) user exit routines.

In this chapter:

- “General User-Supplied Exit Routine Interface Information for CQS”
- “CQS Initialization-Termination User-Supplied Exit Routine” on page 52
- “CQS Client Connection User-Supplied Exit Routine” on page 53
- “Queue Overflow User-Supplied Exit Routine for CQS” on page 55
- “CQS Structure Statistics User-Supplied Exit Routine” on page 57
- “CQS Structure Event User-Supplied Exit Routine” on page 66
- “CQS Statistics Available through the BPE Statistics User Exit” on page 71

This chapter contains Product-sensitive Programming Interface information.

CQS user exit routines enable you to customize and monitor your CQS environment. You write these exit routines, no samples are provided. The CQS user exit routines receive control in the CQS address space in an authorized state. CQS uses Base Primitive Environment (BPE) services to call and manage the CQS user exit routines.

Following is a list of the user exit routines and their functions:

**CQS Initialization-Termination**
Called during CQS initialization and CQS normal termination.

**CQS Client Connection**
Called when a client connects to or disconnects from a structure.

**CQS Queue Overflow**
Called during overflow processing to verify queue name eligibility for overflow processing.

**CQS Structure Statistics**
Called at the end of CQS system checkpoint to allow structure-related statistics gathering.

**CQS Structure Event**
Called during processing for structure processing-related event notification.

In addition, you can use the BPE Statistics User exit to gather CQS statistics; for more information see “CQS Statistics Available through the BPE Statistics User Exit” on page 71

---

General User-Supplied Exit Routine Interface Information for CQS

CQS uses BPE services to call and manage its user exit routines. BPE gives you the ability to externally specify the user exit routine modules to be called for a particular exit routine type via EXITDEF= statements in BPE user exit PROCLIB members. BPE also provides a common user exit routine execution environment. This environment includes:

- Standard BPE user exit parameter list
Recommendation: Write CQS user exit routines in assembler, not in a high level language. CQS does not support exit routines running under Language Environment® for MVS. If you write an exit routine in a high level language, and that routine is executing in the Language Environment for MVS, you might have abends or performance problems. Language Environment for MVS is designed for applications running in key 8, problem program state. CQS user exit routines execute in key 7 supervisor state.

Related Reading
- For complete information about displaying and refreshing user exits, see IMS Version 8: Base Primitive Environment Guide and Reference.
- For complete information about BPE interfaces and services that are available to user exits, see IMS Version 8: Base Primitive Environment Guide and Reference.

CQS Initialization-Termination User-Supplied Exit Routine

The Initialization-Termination (Init-Term) exit routine is called during CQS initialization and CQS normal termination. The Init-Term exit routine is not called during CQS abnormal termination. This exit routine is optional.

The CQS Init-Term user exit routine is driven for the following events:
- CQS initialization; after CQS has completed its initial processing, but before it connects to any structures.
- CQS normal termination, during CQS address space termination, after CQS has disconnected from all structures.

The Init-Term exit routine is defined as TYPE = INITTERM in the EXITDEF statement in the BPE user exit PROCLIB member. You can specify one or more user exit routines of this type. When this exit routine is invoked, the exit routines are driven in the order they are specified by the EXITS= keyword.

Recommendation: Write the Init-Term exit routine so that it is reentrant. It is invoked AMODE 31.

Contents of Registers on Entry

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of Standard BPE user exit parameter list (mapped by the BPEUXPL macro). See IMS Version 8: Base Primitive Environment Guide and Reference for more information.</td>
</tr>
<tr>
<td>13</td>
<td>Address of two pre-chained save areas. The first save area can be used by the exit routine to save registers on entry. The second save area can be used by routines that are called from the user exit routine.</td>
</tr>
<tr>
<td>14</td>
<td>Return address.</td>
</tr>
<tr>
<td>15</td>
<td>Entry point of the exit routine.</td>
</tr>
</tbody>
</table>
Contents of Registers on Exit

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Always set this to zero.

All other registers must be restored.

CQS Initialization and Termination Parameter Lists

On entry to the Init-Term exit routine R1 points to a Standard BPE user exit parameter list. The field UXPL_EXITPLP in this list contains the address of the Init-Term user exit routine parameter lists (mapped by the CQSINTMX macro). The parameters are described in Table 3 and in Table 4.

Table 3. CQS Init-Term User-Supplied Exit Routine Parameter List: CQS Initialization

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITXPVSN</td>
<td>X'00'</td>
<td>X'04'</td>
<td>Input</td>
<td>Parameter List Version Number (00000001)</td>
</tr>
<tr>
<td>ITXFUNC</td>
<td>X'04'</td>
<td>X'04'</td>
<td>Input</td>
<td>Function code</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 CQS Initialization (ITXFINIT)</td>
</tr>
<tr>
<td>ITXCQSID</td>
<td>X'08'</td>
<td>X'08'</td>
<td>Input</td>
<td>CQS identifier</td>
</tr>
<tr>
<td>ITXCQSVN</td>
<td>X'10'</td>
<td>X'04'</td>
<td>Input</td>
<td>CQS version number</td>
</tr>
</tbody>
</table>

Table 4. CQS Init-Term User-Supplied Exit Routine Parameter List: CQS Termination

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITXPVSN</td>
<td>X'00'</td>
<td>X'04'</td>
<td>Input</td>
<td>Parameter List Version Number (00000001)</td>
</tr>
<tr>
<td>ITXFUNC</td>
<td>X'04'</td>
<td>X'04'</td>
<td>Input</td>
<td>Function code</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2 CQS Normal Termination (ITXFNTRM)</td>
</tr>
<tr>
<td>ITXCQSID</td>
<td>X'08'</td>
<td>X'08'</td>
<td>Input</td>
<td>CQS identifier</td>
</tr>
<tr>
<td>ITXCQSVN</td>
<td>X'10'</td>
<td>X'04'</td>
<td>Input</td>
<td>CQS version number</td>
</tr>
</tbody>
</table>

CQS Client Connection User-Supplied Exit Routine

This exit routine is called when a client connects to or disconnects from a structure. This exit routine is optional.

The Client Connection exit routine is driven for the following events:
• Client connect; after a client successfully connects to one or more structures.
• Client disconnect; after a client disconnects normally or abnormally from one or more structures.

The Client Connection exit routine is defined as TYPE = CLNTCONN in the EXITDEF statement in the BPE user exit PROCLIB member. You can specify one or more user exit routines of this type. When this exit routine is invoked, all user exit routines of this type are driven in the order specified by the EXITS= keyword.

Recommendation: Write the Client Connection exit routine so that it is reentrant. It is invoked AMODE 31.
Client Connection User Exit

Contents of Registers on Entry

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of Standard BPE user exit parameter list (mapped by the BPEUXPL macro). See <a href="#">IMS Version 8: Base Primitive Environment Guide and Reference</a> for more information.</td>
</tr>
<tr>
<td>13</td>
<td>Address of two pre-chained save areas. The first save area can be used by the exit routine to save registers on entry. The second save area can be used by routines that are called from the user exit routine.</td>
</tr>
<tr>
<td>14</td>
<td>Return address.</td>
</tr>
<tr>
<td>15</td>
<td>Entry point of the exit routine.</td>
</tr>
</tbody>
</table>

Contents of Registers on Exit

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
<tr>
<td></td>
<td>0         Always set this to zero.</td>
</tr>
</tbody>
</table>

All other registers must be restored.

CQS Client Connection and Disconnect Parameter Lists

On entry to the Client Connection exit routine, R1 points to a Standard BPE user exit parameter list. The field UXPL_EXITLP in this list contains the address of the Client Connection user exit routine parameter list (mapped by the CQSCLNCX macro). The parameters for client connection are described in Table 5. The parameters for client disconnect are described in Table 6.

Table 5. CQS Client Connection User-Supplied Exit Routine Parameter List: Client Connection

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCXPVSN</td>
<td>X’00’</td>
<td>X’04’</td>
<td>Input</td>
<td>Parameter List Version Number (00000001).</td>
</tr>
<tr>
<td>CCXFUNC</td>
<td>X’04’</td>
<td>X’04’</td>
<td>Input</td>
<td>Function code</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Client Connect (CCXFCONN).</td>
</tr>
<tr>
<td>CCXCQSID</td>
<td>X’08’</td>
<td>X’08’</td>
<td>Input</td>
<td>CQS identifier.</td>
</tr>
<tr>
<td>CCXCQSVN</td>
<td>X’10’</td>
<td>X’04’</td>
<td>Input</td>
<td>CQS version number.</td>
</tr>
<tr>
<td>CCXCLNNM</td>
<td>X’14’</td>
<td>X’08’</td>
<td>Input</td>
<td>Client name.</td>
</tr>
<tr>
<td>CCXCSNUM</td>
<td>X’1C’</td>
<td>X’04’</td>
<td>Input</td>
<td>Number of structure name entries in the list.</td>
</tr>
<tr>
<td>CCXCSENL</td>
<td>X’20’</td>
<td>X’04’</td>
<td>Input</td>
<td>Length of each structure name list entry.</td>
</tr>
<tr>
<td>CCXCSLST</td>
<td>X’24’</td>
<td>X’04’</td>
<td>Input</td>
<td>Address of first structure name entry. Each entry contains the 16-byte name of a structure that the client connected to.</td>
</tr>
</tbody>
</table>

Table 6. CQS Client Connection User-Supplied Exit Routine Parameter List: Client Disconnect

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCXPVSN</td>
<td>X’00’</td>
<td>X’04’</td>
<td>Input</td>
<td>Parameter List Version Number (00000001).</td>
</tr>
<tr>
<td>CCXFUNC</td>
<td>X’04’</td>
<td>X’04’</td>
<td>Input</td>
<td>Function code</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>Client Disconnect (CCXFDISC).</td>
</tr>
<tr>
<td>CCXCQSID</td>
<td>X’08’</td>
<td>X’08’</td>
<td>Input</td>
<td>CQS identifier.</td>
</tr>
<tr>
<td>CCXCQSVN</td>
<td>X’10’</td>
<td>X’04’</td>
<td>Input</td>
<td>CQS version number.</td>
</tr>
</tbody>
</table>
Table 6. CQS Client Connection User-Supplied Exit Routine Parameter List: Client Disconnect (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCXCLNNM</td>
<td>X'14'</td>
<td>X'08'</td>
<td>Input</td>
<td>Client name.</td>
</tr>
<tr>
<td>CCXDFLG1</td>
<td>X'1C'</td>
<td>X'01'</td>
<td>Input</td>
<td>Flag byte indicates whether the client disconnect is abnormal X'80' Client disconnect is abnormal (CCXDABND).</td>
</tr>
<tr>
<td>N/A</td>
<td>X'1D'</td>
<td>X'03'</td>
<td></td>
<td>Reserved.</td>
</tr>
<tr>
<td>CCXDSNUM</td>
<td>X'20'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of structure name entries in the list.</td>
</tr>
<tr>
<td>CCXDSENL</td>
<td>X'24'</td>
<td>X'04'</td>
<td>Input</td>
<td>Length of each structure name list entry.</td>
</tr>
<tr>
<td>CCXDSLST</td>
<td>X'28'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of first structure name entry. Each entry contains the 16-byte name of a structure that the client disconnected from.</td>
</tr>
</tbody>
</table>

Queue Overflow User-Supplied Exit Routine for CQS

The Queue Overflow exit routine is called during overflow queue selection processing to approve or veto a queue name for overflow processing.

This exit routine is optional.

During overflow processing the Queue Overflow exit routine is called to verify that a queue name selected by CQS is eligible for overflow processing. When CQS determines that the structure has reached its overflow threshold, overflow threshold processing begins. Then CQS determines which queues are using the most storage in the structure. The queues using the most storage in the structure become candidates for overflow and are moved to the overflow structure. Or, if no overflow structure is defined, the queues using the most storage in the structure no longer allow CQSPUT requests for the queue.

Restriction: The queue overflow user exit does not apply to the resource structure.

During queue selection processing the Queue Overflow exit routine is invoked once per selected queue name to approve or veto the queue name for overflow processing. If the exit routine approves the move or the exit routine is not specified, all data objects for that queue (such as IMS messages for that destination) are moved to the overflow structure. All additional processing for that queue name is done in the overflow structure, if the overflow structure exists. If no overflow structure exists, CQSPUT requests to the queue are rejected. If the move is vetoed, the queue name is removed from the overflow candidate list, and another queue name is selected.

The Queue Overflow exit routine is defined as TYPE = OVERFLOW in the EXITDEF statement in the BPE user exit PROCLIB member. You can specify one or more user exit routines of this type. When this exit routine is invoked, all such routines are driven in the order specified by the EXITS= keyword.

Because multiple overflow exit routines might exist, the last exit routine called is the one that determines whether the queue name is selected for overflow. If an exit routine accepts a queue name as one that is valid for overflow processing or does not recognize the name, the exit routine must set R15 to 0 and specify that the next exit in the list should be called. This allows the next exit routine to have a chance to veto the name selection. If an exit routine determines that a queue name is ineligible as a candidate for overflow processing, the exit routine must set R15 to 4 and specify that no more exit routines are to be called.
Queue Overflow User Exit

Within the Standard BPE user exit parameter list is the field UXPL_CALLNEXTP, which is a pointer to a byte of storage which is set by the exit routine to indicate whether the next exit routine in the list is to be called. When the byte of storage is set to UXPL_CALLNEXTYES, the next exit is called (if one exists). When the byte of storage is set to UXPL_CALLNEXTNO, no more exits are called for this queue name.

If a Queue Overflow exit routine determines that a queue name is not a candidate for overflow, the exit routine can set the byte pointed to by field UXPL_CALLNEXTP to the value of UXPL_CALLNEXTNO (X'04') so that no other exit routines are called for the queue name.

Recommendation: Write the Queue Overflow exit routine so that it is reentrant. It is invoked AMODE 31.

Contents of Registers on Entry

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of Standard BPE user exit parameter list (mapped by the BPEUXPL macro). See <a href="https://www.ibm.com">IMS Version 8: Base Primitive Environment Guide and Reference</a> for more information.</td>
</tr>
<tr>
<td>13</td>
<td>Address of two pre-chained save areas. The first save area can be used by the exit routine to save registers on entry. The second save area can be used by routines that are called from the user exit routine.</td>
</tr>
<tr>
<td>14</td>
<td>Return address.</td>
</tr>
<tr>
<td>15</td>
<td>Entry point of the exit routine.</td>
</tr>
</tbody>
</table>

Contents of Registers on Exit

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
<tr>
<td>0</td>
<td>Allow queue to be moved to overflow structure.</td>
</tr>
<tr>
<td>4</td>
<td>Do not move queue to overflow structure; select another candidate.</td>
</tr>
</tbody>
</table>

Attention: This return code is ignored unless the exit routine is the last overflow user exit called for the queue name.

An exit routine is considered the last one called when either of the following are true:

1. The exit routine is the last routine defined in the exit list for the overflow queue.
2. The exit routine sets the byte pointed to by UXPL_CALLNEXTP to the value UXPL_CALLNEXTNO.

All other registers must be restored.

CQS Queue Overflow Parameter List

On entry to the Queue Overflow exit routine, R1 points to a Standard BPE user exit parameter list. The field UXPL_EXITPLP in this list contains the address of the CQS
Queue Overflow user exit routine parameter list (mapped by the CQSQOFLX macro). The parameters are described in detail in Table 7.

Table 7. CQS Queue Overflow User-Supplied Exit Routine Parameter List

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QOXPVSN</td>
<td>X'00'</td>
<td>X'04'</td>
<td>Input</td>
<td>Parameter List Version Number (00000001).</td>
</tr>
<tr>
<td>QOXFUNC</td>
<td>X'04'</td>
<td>X'04'</td>
<td>Input</td>
<td>Function code 1 Queue Name Selection (QOXFQOFL).</td>
</tr>
<tr>
<td>QOXQOFL1</td>
<td>X'08'</td>
<td>X'01'</td>
<td>Input</td>
<td>Flag byte indicating whether this is the first overflow exit call for this overflow threshold process. The exit routine is called once per selected queue name for each occurrence of overflow threshold processing. This bit will be on for the first queue name for an occurrence of overflow threshold processing. X'80' This is the initial entry for this overflow threshold process (QOXQ1ST).</td>
</tr>
<tr>
<td>N/A</td>
<td>X'09'</td>
<td>X'03'</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>QOXCQSID</td>
<td>X'0C'</td>
<td>X'08'</td>
<td>Input</td>
<td>CQS identifier.</td>
</tr>
<tr>
<td>QOXCQSVN</td>
<td>X'14'</td>
<td>X'04'</td>
<td>Input</td>
<td>CQS version number.</td>
</tr>
<tr>
<td>QOXSTRNM</td>
<td>X'18'</td>
<td>X'10'</td>
<td>Input</td>
<td>Structure Name.</td>
</tr>
<tr>
<td>QOXQNAME</td>
<td>X'28'</td>
<td>X'10'</td>
<td>Input</td>
<td>Queue name selected for overflow processing.</td>
</tr>
<tr>
<td>QOXDOBJN</td>
<td>X'38'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of data objects on the selected queue name.</td>
</tr>
</tbody>
</table>

CQS Structure Statistics User-Supplied Exit Routine

The CQS Structure Statistics user exit routine enables you to gather statistics related to the structure. This exit routine is optional.

The exit routine is driven at the end of a successful system checkpoint. All statistical data that CQS gathers, including rebuild statistics and checkpoint statistics, are passed to the Structure Statistics user exit at the end of each successful system checkpoint. All statistical data is logged in the Structure Statistics log record. You can also obtain this same statistical data with the CQSQUERY FUNC=STRSTAT request.

**Recommendation:** Some statistics about resource structures are passed in the structure statistics. CQS system checkpoint does not apply to resource structures. Use the STATINV parameter in the BPE configuration PROCLIB member to define the time interval so that BPE regularly drives CQS’s statistics user exit. See the IMS Version 8: Base Primitive Environment Guide and Reference for more information about the BPE configuration PROCLIB member.

The CQS Structure Statistics user exit routine is defined as TYPE = STRSTAT in the EXITDEF statement in the BPE user exit PROCLIB member. You can specify one or more user exit routines of this type. When this exit routine is invoked, all routines of this type are driven in the order specified by the EXITS= keyword.

**Recommendation:** Write the CQS Structure Statistics exit routine so that it is reentrant. It is invoked AMODE 31.

Contents of Registers on Entry

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
</table>

Chapter 4. CQS User-Supplied Exit Routines 57
Structure Statistics User Exit

1 Address of Standard BPE user exit parameter list (mapped by the BPEUXPL macro). See IMS Version 8 Base Primitive Guide and Reference for more information.

13 Address of two pre-chained save areas. The first save area can be used by the exit routine to save registers on entry. The second save area can be used by routines that are called from the user exit routine.

14 Return address.

15 Entry point of the exit routine.

Contents of Registers on Exit

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
<tr>
<td></td>
<td>0 Always set this to zero.</td>
</tr>
</tbody>
</table>

All other registers must be restored.

CQS Structure Statistics User-Supplied Exit Routine Parameter List

On entry to the Structure Statistics exit routine, R1 points to a Standard BPE user exit parameter list. The field UXPL_EXITPLP in this list contains the address of the CQS Structure Statistics user exit routine parameter list (mapped by the CQSSSTATX macro). The parameters are described in Table 8.

Table 8. CQS Structure Statistics User-Supplied Exit Routine Parameter List

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXPVSN</td>
<td>X'00'</td>
<td>X'04'</td>
<td>Input</td>
<td>Parameter List Version Number (00000001).</td>
</tr>
<tr>
<td>SAXFUNC</td>
<td>X'04'</td>
<td>X'04'</td>
<td>Input</td>
<td>Function code</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 System Checkpoint (SAXFCSYS).</td>
</tr>
<tr>
<td>SAXCQSID</td>
<td>X'08'</td>
<td>X'08'</td>
<td>Input</td>
<td>CQS identifier.</td>
</tr>
<tr>
<td>SAXCQSVN</td>
<td>X'10'</td>
<td>X'04'</td>
<td>Input</td>
<td>CQS version number.</td>
</tr>
<tr>
<td>SAXSTRNM</td>
<td>X'14'</td>
<td>X'10'</td>
<td>Input</td>
<td>Structure name.</td>
</tr>
<tr>
<td>SAXSSTT1</td>
<td>X'24'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of structure process statistics record for activity performed by</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CQS processes on this structure for all clients since restart or the last</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>successful structure checkpoint (mapped by the CQSSSTT1 macro). See Table 9 on page 59 for a description of the process statistics record.</td>
</tr>
<tr>
<td>SAXSSTT2</td>
<td>X'28'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of CQS request statistics record for activity performed for CQS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>requests for this structure for all clients since restart or the last</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>successful structure checkpoint (mapped by the CQSSSTT2 macro). See Table 10 on page 59 for a description of the request statistics record.</td>
</tr>
<tr>
<td>SAXSSTT3</td>
<td>X'2C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of data object statistics record for activity performed on data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>objects in this structure for all clients since restart or the last</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>successful structure checkpoint (mapped by the CQSSSTT3 macro). See Table 11 on page 60 for a description of the object statistics record.</td>
</tr>
<tr>
<td>SAXSSTT4</td>
<td>X'30'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of queue name statistics record for activity performed on queue</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>names in this structure for all clients since restart or the last</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>successful structure checkpoint (mapped by the CQSSSTT4 macro). See Table 12 on page 61 for a description of the queue name statistics record.</td>
</tr>
</tbody>
</table>

58 Common Queue Server Guide and Reference
Table 8. CQS Structure Statistics User-Supplied Exit Routine Parameter List (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAXSSTT5</td>
<td>X'34'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of MVS request statistics record for activity performed by CQS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>processes on this structure for all clients since restart or the last</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>successful structure checkpoint (mapped by the CQSSSTT5 macro). See Table 13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>on page 61 for a description of the MVS request statistics record.</td>
</tr>
<tr>
<td>SAXSSTT6</td>
<td>X'38'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of rebuild statistics record containing data from the last</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>rebuild in which this CQS acted as master (mapped by the CQSSSTT6 macro).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>See Table 14 on page 62 for a description of the rebuild statistics record.</td>
</tr>
<tr>
<td>SAXSSTT7</td>
<td>X'3C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of structure checkpoint statistics record containing data from</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>the last three structure checkpoints in which this CQS acted as master</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(mapped by the CQSSSTT7 macro). See Table 15 on page 65 for a description</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>of the structure checkpoint statistics record.</td>
</tr>
</tbody>
</table>

CQS Structure Process Statistics Record

Table 9. CQS Structure Process Statistics Record

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS1ID</td>
<td>X'00'</td>
<td>X'08'</td>
<td>Input</td>
<td>Eye-catcher CQSSSTT1</td>
</tr>
<tr>
<td>SS1LN</td>
<td>X'08'</td>
<td>X'04'</td>
<td>Input</td>
<td>Length of valid data</td>
</tr>
<tr>
<td>SS1PVSN</td>
<td>X'0C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Parameter List Version Number (00000002)</td>
</tr>
<tr>
<td>SS1STATS</td>
<td>X'10'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of times CQS successfully performed system checkpoint processing for</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>the structure</td>
</tr>
<tr>
<td>SS1TCHKP</td>
<td>X'14'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of times CQS successfully performed structure checkpoint processing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>for the structure</td>
</tr>
<tr>
<td>SS1RBLD</td>
<td>X'18'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of times CQS successfully performed rebuild processing for the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>structure</td>
</tr>
<tr>
<td>SS1DUPLX</td>
<td>X'20'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of times CQS successfully established a duplexing rebuild</td>
</tr>
<tr>
<td>SS1OFLWT</td>
<td>X'1C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of times CQS performed overflow threshold processing for the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>structure</td>
</tr>
</tbody>
</table>

CQS Request Statistics Record

Table 10. CQS Request Statistics Record

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS2ID</td>
<td>X'00'</td>
<td>X'08'</td>
<td>Input</td>
<td>Eyecatcher CQSSSTT2</td>
</tr>
<tr>
<td>SS2LN</td>
<td>X'08'</td>
<td>X'04'</td>
<td>Input</td>
<td>Length of valid data</td>
</tr>
<tr>
<td>SS2PVSN</td>
<td>X'0C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Parameter List Version Number (00000002)</td>
</tr>
<tr>
<td>SS2BRWSE</td>
<td>X'10'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of COSBRWSE requests for this structure</td>
</tr>
<tr>
<td>SS2CHKPT</td>
<td>X'14'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of COSCHKPT requests for this structure</td>
</tr>
<tr>
<td>SS2CONN</td>
<td>X'18'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of COSCONN requests for this structure</td>
</tr>
<tr>
<td>SS2DEL</td>
<td>X'1C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of COSDEL requests for this structure</td>
</tr>
<tr>
<td>SS2DISC</td>
<td>X'20'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of COSDISC requests for this structure</td>
</tr>
<tr>
<td>SS2INFRM</td>
<td>X'24'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of COSINFRM requests for this structure</td>
</tr>
</tbody>
</table>
### Structure Statistics User Exit

Table 10. CQS Request Statistics Record (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS2MOVE</td>
<td>X'28'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of CQSMOVE requests for this structure</td>
</tr>
<tr>
<td>SS2PUT</td>
<td>X'2C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of CQSPUT requests for this structure</td>
</tr>
<tr>
<td>SS2QUERY</td>
<td>X'30'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of CQSQUERY requests for this structure</td>
</tr>
<tr>
<td>SS2READ</td>
<td>X'34'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of CQSREAD requests for this structure</td>
</tr>
<tr>
<td>SS2RECVR</td>
<td>X'38'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of CQSRECVR requests for this structure</td>
</tr>
<tr>
<td>SS2SYNC</td>
<td>X'3C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of CQSSYNC requests for this structure</td>
</tr>
<tr>
<td>SS2UNLCK</td>
<td>X'40'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of CQSUNLCK requests for this structure</td>
</tr>
<tr>
<td>SS2UPD</td>
<td>X'44'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of CQSUPD requests for this structure</td>
</tr>
</tbody>
</table>

### Data Object Statistics Record for CQS

Table 11 describes the Structure Statistics user exit routine data object statistics record.

Table 11. Data Object Statistics Record

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS3ID</td>
<td>X'00'</td>
<td>X'08'</td>
<td>Input</td>
<td>Eyecatcher CQSSSTT3.</td>
</tr>
<tr>
<td>SS3LN</td>
<td>X'08'</td>
<td>X'04'</td>
<td>Input</td>
<td>Length of valid data.</td>
</tr>
<tr>
<td>SS3PVSN</td>
<td>X'0C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Parameter List Version Number (00000002).</td>
</tr>
<tr>
<td>SS3PTOBJ</td>
<td>X'10'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of data objects added to the structure with COMMIT = NO. This count does not include data objects added with COMMIT = YES or RECOVERABLE = NO.</td>
</tr>
<tr>
<td>SS3PTCMT</td>
<td>X'14'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of data objects added to the structure with COMMIT = YES. This count indicates the number of recoverable UOWs added to the structure. This count plus the number of data objects that are added with COMMIT = NO is the total number of recoverable data objects added to the structure.</td>
</tr>
<tr>
<td>SS3PTNRO</td>
<td>X'18'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of data objects added to the structure with RECOVERABLE = NO. This count indicates the number of nonrecoverable UOWs added to the structure. This count plus the number of data objects that are added with COMMIT = YES is the total number of UOWs that were added to the structure.</td>
</tr>
<tr>
<td>SS3RDOBJ</td>
<td>X'1C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of data objects read from the structure.</td>
</tr>
<tr>
<td>SS3MVOBJ</td>
<td>X'20'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of data objects moved from one queue to another on the structure.</td>
</tr>
<tr>
<td>SS3ULOBJ</td>
<td>X'24'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of data objects unlocked on the structure.</td>
</tr>
<tr>
<td>SS3ENTAL</td>
<td>X'30'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of data entries allocated on the structure. Compare the data entry in use field to the data entry allocated field to determine how close the structure is to becoming full.</td>
</tr>
<tr>
<td>SS3ENTIN</td>
<td>X'34'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of data entries in use on the structure. Compare the data entry in use field to the data entry allocated field to determine how close the structure is to becoming full.</td>
</tr>
<tr>
<td>SS3ENTHI</td>
<td>X'38'</td>
<td>X'04'</td>
<td>Input</td>
<td>High water mark for number of data entries on the structure. Compare the data entry in use field to the data entry allocated field to determine how close the structure is to becoming full.</td>
</tr>
<tr>
<td>SS3ENTTM</td>
<td>X'3C'</td>
<td>X'08'</td>
<td>Input</td>
<td>Timestamp representing the time the data entry high water mark was reached for the structure (in STCK format).</td>
</tr>
<tr>
<td>SS3ELMAL</td>
<td>X'44'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of data elements allocated on the structure. Compare the data entry in use field to the data entry allocated field to determine how close the structure is to becoming full.</td>
</tr>
</tbody>
</table>
Table 11. Data Object Statistics Record (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS3ELMIN</td>
<td>X'48'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of data elements in use on the structure. Compare the data entry in use field to the data entry allocated field to determine how close the structure is to becoming full.</td>
</tr>
<tr>
<td>SS3ELMHI</td>
<td>X'4C'</td>
<td>X'04'</td>
<td>Input</td>
<td>High water mark for number of data elements on the structure. Compare the data element high water mark field to the data element allocated field to determine the closest the structure came to becoming full.</td>
</tr>
<tr>
<td>SS3ELMTM</td>
<td>X'50'</td>
<td>X'04'</td>
<td>Input</td>
<td>Timestamp representing the time the data element high water mark was reached for the structure (in STCK format).</td>
</tr>
<tr>
<td>Reserved</td>
<td>X'58'</td>
<td>X'04'</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>X'5C'</td>
<td>X'04'</td>
<td>Input</td>
<td></td>
</tr>
</tbody>
</table>

Queue Name Statistics Record for CQS

Table 12 describes the Structure Statistics user exit routine queue name statistics record.

Restriction: The queue name statistics record does not apply to resource structures.

Table 12. Queue Name Statistics Record

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS4ID</td>
<td>X'00'</td>
<td>X'08'</td>
<td>Input</td>
<td>Eyecatcher CQSSSTT4</td>
</tr>
<tr>
<td>SS4LN</td>
<td>X'08'</td>
<td>X'04'</td>
<td>Input</td>
<td>Length of valid data</td>
</tr>
<tr>
<td>SS4PVSN</td>
<td>X'0C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Parameter List Version Number (00000001)</td>
</tr>
<tr>
<td>SS4NFQN</td>
<td>X'10'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of queue names for which an inform was performed</td>
</tr>
<tr>
<td>SS4UNFQN</td>
<td>X'14'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of queue names for which an uninform was performed</td>
</tr>
<tr>
<td>SS4NFYQN</td>
<td>X'18'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of queue name notifications (when a queue goes from empty to non-empty)</td>
</tr>
</tbody>
</table>

z/OS Request Statistics Record for CQS

Table 13 describes the Structure Statistics user exit routine MVS request statistics record.

Table 13. MVS Request Statistics Record

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS5ID</td>
<td>X'00'</td>
<td>X'08'</td>
<td>Input</td>
<td>Eyecatcher CQSSSTT5.</td>
</tr>
<tr>
<td>SS5LN</td>
<td>X'08'</td>
<td>X'04'</td>
<td>Input</td>
<td>Length of valid data.</td>
</tr>
<tr>
<td>SS5PVSN</td>
<td>X'0C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Parameter List Version Number (00000002).</td>
</tr>
<tr>
<td>SS5IXGWR</td>
<td>X'10'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of IXGWRITE requests for the structure. This represents the number of log records written during processing on the structure.</td>
</tr>
<tr>
<td>SS5IXGBR</td>
<td>X'14'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of IXGBRWSE requests for the structure.</td>
</tr>
<tr>
<td>SS5IXLDO</td>
<td>X'18'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of IXLLIST DEQ_EVENTQ requests for the structure.</td>
</tr>
<tr>
<td>SS5IXLWR</td>
<td>X'1C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of IXLLIST WRITE requests for the structure.</td>
</tr>
<tr>
<td>SS5IXLRC</td>
<td>X'20'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of IXLLIST READ requests for the structure.</td>
</tr>
<tr>
<td>SS5IXLMV</td>
<td>X'24'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of IXLLIST MOVE requests for the structure.</td>
</tr>
<tr>
<td>SS5IXLDL</td>
<td>X'28'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of IXLLIST DELETE requests for the structure.</td>
</tr>
</tbody>
</table>
Structure Statistics User Exit

Table 13. MVS Request Statistics Record (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS5IXLMG</td>
<td>X'2C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of IXLMG requests for the structure.</td>
</tr>
<tr>
<td>SS5IXLUS</td>
<td>X'30'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of IXLUSYNC requests for the structure.</td>
</tr>
<tr>
<td>SS5IXEWR</td>
<td>X'34'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of IXLLSTE WRITE requests for the structure.</td>
</tr>
<tr>
<td>SS5IXERD</td>
<td>X'38'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of IXLLSTE READ requests for the structure.</td>
</tr>
<tr>
<td>SS5IXMRL</td>
<td>X'3C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of IXLLSTM READ_LIST requests for the structure.</td>
</tr>
<tr>
<td>SS5IXEDL</td>
<td>X'40'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of IXLLSTE DELETE requests for the structure.</td>
</tr>
<tr>
<td>SS5IXMDL</td>
<td>X'44'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of IXLLSTM DELETE_ENTRYLIST requests for the structure.</td>
</tr>
</tbody>
</table>

Structure Rebuild Statistics Record for CQS

Structure rebuild statistics are gathered only by the CQS that is the master of the structure rebuild process. A CQS has access only to the data it gathers. Each CQS keeps structure rebuild statistics for the last rebuild for which it was the master.

Table 14 describes the Structure Statistics user exit routine structure rebuild statistics record.

Table 14. Structure Rebuild Statistics Record

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS6ID</td>
<td>X'00'</td>
<td>X'08'</td>
<td>Input</td>
<td>Eyecatcher CQSSSTT6.</td>
</tr>
<tr>
<td>SS6LN</td>
<td>X'08'</td>
<td>X'04'</td>
<td>Input</td>
<td>Length of valid data.</td>
</tr>
<tr>
<td>SS6PVSN</td>
<td>X'0C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Parameter List Version Number (00000003).</td>
</tr>
<tr>
<td>SS6ELMIO</td>
<td>X'10'</td>
<td>X'04'</td>
<td>Input</td>
<td>Data elements in use on old structure.</td>
</tr>
<tr>
<td>SS6ELMAO</td>
<td>X'14'</td>
<td>X'04'</td>
<td>Input</td>
<td>Data elements allocated on old structure.</td>
</tr>
<tr>
<td>SS6ENTIO</td>
<td>X'18'</td>
<td>X'04'</td>
<td>Input</td>
<td>Data entries in use on old structure (data object count).</td>
</tr>
<tr>
<td>SS6ENTAO</td>
<td>X'1C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Data entries allocated on old structure.</td>
</tr>
<tr>
<td>SS6MCIO</td>
<td>X'20'</td>
<td>X'04'</td>
<td>Input</td>
<td>EMCs in use on old structure (active informs).</td>
</tr>
<tr>
<td>SS6EMCAO</td>
<td>X'24'</td>
<td>X'04'</td>
<td>Input</td>
<td>EMCs in use on old structure (active informs).</td>
</tr>
<tr>
<td>SS6SIZEO</td>
<td>X'28'</td>
<td>X'04'</td>
<td>Input</td>
<td>Old structure size in 4K blocks.</td>
</tr>
<tr>
<td>SS6CFTO</td>
<td>X'2C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Old CF total space in 4K blocks.</td>
</tr>
<tr>
<td>SS6CFFO</td>
<td>X'30'</td>
<td>X'04'</td>
<td>Input</td>
<td>Old CF free space in 4K blocks.</td>
</tr>
<tr>
<td>SS6CFNMO</td>
<td>X'34'</td>
<td>X'08'</td>
<td>Input</td>
<td>Old CF name in which structure was allocated before rebuild.</td>
</tr>
<tr>
<td></td>
<td>X'3C'</td>
<td>X'04'</td>
<td></td>
<td>Unused.</td>
</tr>
<tr>
<td>SS6ELMIN</td>
<td>X'40'</td>
<td>X'04'</td>
<td>Input</td>
<td>Data elements in use on new structure.</td>
</tr>
<tr>
<td>SS6ELMAN</td>
<td>X'44'</td>
<td>X'04'</td>
<td>Input</td>
<td>Data elements allocated on new structure.</td>
</tr>
<tr>
<td>SS6ENTIN</td>
<td>X'48'</td>
<td>X'04'</td>
<td>Input</td>
<td>Data entries in use on new structure (data object count).</td>
</tr>
<tr>
<td>SS6ENTAN</td>
<td>X'4C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Data entries allocated on new structure.</td>
</tr>
<tr>
<td>SS6EMCIN</td>
<td>X'50'</td>
<td>X'04'</td>
<td>Input</td>
<td>EMCs in use on new structure (active informs).</td>
</tr>
<tr>
<td>SS6EMCAN</td>
<td>X'54'</td>
<td>X'04'</td>
<td>Input</td>
<td>EMCs in use on new structure (active informs).</td>
</tr>
<tr>
<td>SS6SIZEN</td>
<td>X'58'</td>
<td>X'04'</td>
<td>Input</td>
<td>New structure size in 4K blocks.</td>
</tr>
<tr>
<td>SS6CFTN</td>
<td>X'5C'</td>
<td>X'04'</td>
<td>Input</td>
<td>New CF total space in 4K blocks.</td>
</tr>
<tr>
<td>SS6CFFN</td>
<td>X'60'</td>
<td>X'04'</td>
<td>Input</td>
<td>New CF free space in 4K blocks.</td>
</tr>
<tr>
<td>SS6CFNIN</td>
<td>X'64'</td>
<td>X'08'</td>
<td>Input</td>
<td>New CF name in which structure is allocated after rebuild.</td>
</tr>
<tr>
<td></td>
<td>X'6C'</td>
<td>X'04'</td>
<td></td>
<td>Unused.</td>
</tr>
<tr>
<td>SS6RBTIM</td>
<td>X'70'</td>
<td>X'08'</td>
<td>Input</td>
<td>Rebuild timestamp (STCK).</td>
</tr>
</tbody>
</table>
Table 14. Structure Rebuild Statistics Record (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS6POPCT</td>
<td>X’78’</td>
<td>X’04’</td>
<td>Input</td>
<td>Repopulation from SRDS count (RCVRY) or objects copied count (COPY).</td>
</tr>
<tr>
<td>SS6MVQCT</td>
<td>X’7C’</td>
<td>X’04’</td>
<td>Input</td>
<td>Entries moved to moveq during phase 2 count.</td>
</tr>
<tr>
<td>SS6PUTCT</td>
<td>X’80’</td>
<td>X’04’</td>
<td>Input</td>
<td>Entries written during phase 3 count.</td>
</tr>
<tr>
<td>SS6MOVCT</td>
<td>X’84’</td>
<td>X’04’</td>
<td>Input</td>
<td>Entries moved during phase 3 count.</td>
</tr>
<tr>
<td>SS6OBJCT</td>
<td>X’88’</td>
<td>X’04’</td>
<td>Input</td>
<td>Data objects affected by recovery count (recoverable and nonrecoverable).</td>
</tr>
<tr>
<td>SS6UOWCT</td>
<td>X’8C’</td>
<td>X’04’</td>
<td>Input</td>
<td>UOWs affected by recovery count (recoverable and nonrecoverable).</td>
</tr>
<tr>
<td>SS6FLAG1</td>
<td>X’90’</td>
<td>X’01’</td>
<td>Input</td>
<td>Flag byte.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X’80’ These statistics are for the last rebuild performed for the structure.</td>
</tr>
<tr>
<td>SS6FLAG2</td>
<td>X’91’</td>
<td>X’01’</td>
<td>Input</td>
<td>Rebuild flag.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Indicates the last rebuild or duplexing rebuild event received that</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>updated these rebuild statistics:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1  Structure rebuild statistics.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2  Duplexing started statistics.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3  Duplexing ended statistics and MVS switched to simplex structure (either</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>old or new structure).</td>
</tr>
<tr>
<td></td>
<td>X’91’</td>
<td>X’03’</td>
<td>Unused</td>
<td></td>
</tr>
</tbody>
</table>

The remaining fields of this table apply to rebuild failures. The CQS0242E message identifies the rebuild failure reason.

The following fields apply to rebuild failures that occurred while rebuild was processing a CQS log record. Use this information to locate the log record in the CQS log to give to an IBM service representative.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS6LGTYPE</td>
<td>X’94’</td>
<td>X’01’</td>
<td>Input</td>
<td>Log record type of log record being processed when rebuild failure occurred.</td>
</tr>
<tr>
<td>SS6LGSUB</td>
<td>X’95’</td>
<td>X’01’</td>
<td>Input</td>
<td>Log record subtype of log record being processed when rebuild failure</td>
</tr>
<tr>
<td>SS6STYPE</td>
<td>X’96’</td>
<td>X’01’</td>
<td>Input</td>
<td>Structure type of log record being processed when rebuild failure occurred.</td>
</tr>
<tr>
<td></td>
<td>X’97’</td>
<td>X’01’</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>SS6LGTIM</td>
<td>X’98’</td>
<td>X’08’</td>
<td>Input</td>
<td>Log record time stamp of log record being processed when rebuild failure</td>
</tr>
<tr>
<td>SS6CQSID</td>
<td>X’A0’</td>
<td>X’08’</td>
<td>Input</td>
<td>CQS ID associated with log record being processed when rebuild failure</td>
</tr>
<tr>
<td>SS6CLNTN</td>
<td>X’A8’</td>
<td>X’08’</td>
<td>Input</td>
<td>Client name associated with log record being processed when rebuild failure</td>
</tr>
<tr>
<td>SS6SRCQ</td>
<td>X’B0’</td>
<td>X’10’</td>
<td>Input</td>
<td>Source client or private queue name associated with log record being</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>processed when rebuild failure occurred.</td>
</tr>
<tr>
<td>SS6DSTQ</td>
<td>X’C0’</td>
<td>X’10’</td>
<td>Input</td>
<td>Destination queue name associated with log record being processed when</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>rebuild failure occurred.</td>
</tr>
<tr>
<td>SS6UOW</td>
<td>X’B0’</td>
<td>X’20’</td>
<td>Input</td>
<td>UOW associated with log record being processed when rebuild failure</td>
</tr>
<tr>
<td>SS6UNIQ1</td>
<td>X’F0’</td>
<td>X’04’</td>
<td>Input</td>
<td>Info unique to log record or rebuild data object entry when rebuild failure</td>
</tr>
<tr>
<td>SS6UNIQ2</td>
<td>X’F4’</td>
<td>X’04’</td>
<td>Input</td>
<td>Info unique to log record or rebuild data object entry when rebuild failure</td>
</tr>
<tr>
<td>SS6UNIQ3</td>
<td>X’F8’</td>
<td>X’04’</td>
<td>Input</td>
<td>Info unique to log record or rebuild data object entry when rebuild failure</td>
</tr>
</tbody>
</table>

The following fields apply to rebuild failures that occurred while rebuild was processing an IXL request to access the structure.
Table 14. Structure Rebuild Statistics Record (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS6IXLMC</td>
<td>X'FC'</td>
<td>X'01'</td>
<td>Input</td>
<td>IXL macro that failed and caused rebuild to fail. See CQSTRACE macro for IXL macro type.</td>
</tr>
<tr>
<td></td>
<td>X'FE'</td>
<td>X'02'</td>
<td></td>
<td>Unused.</td>
</tr>
<tr>
<td>SS6IXLRQ</td>
<td>X'FD'</td>
<td>X'01'</td>
<td>Input</td>
<td>IXL request that failed and caused the rebuild to fail.</td>
</tr>
<tr>
<td></td>
<td>X'100'</td>
<td>X'04'</td>
<td></td>
<td>IXL return code returned by IXL request that caused rebuild to fail.</td>
</tr>
<tr>
<td>SS6IXLRN</td>
<td>X'104'</td>
<td>X'04'</td>
<td>Input</td>
<td>IXL reason code returned by IXL request that caused rebuild to fail.</td>
</tr>
<tr>
<td>SS6SRVRC</td>
<td>X'108'</td>
<td>X'04'</td>
<td>Input</td>
<td>This field applies to rebuild failures that occurred while rebuild was processing a service (for example, CQSTBL, BPELAGET, BPECBGET). It provides the return code of the service that failed.</td>
</tr>
<tr>
<td></td>
<td>X'10C'</td>
<td>X'04'</td>
<td></td>
<td>Unused.</td>
</tr>
<tr>
<td>SS6VRSNO</td>
<td>X'110'</td>
<td>X'08'</td>
<td>Input</td>
<td>Old structure version (rebuild) or primary structure version (duplexing rebuild).</td>
</tr>
<tr>
<td>SS6VRSNN</td>
<td>X'118'</td>
<td>X'08'</td>
<td>Input</td>
<td>New structure version (rebuild) or secondary structure version (duplexing rebuild).</td>
</tr>
<tr>
<td>SS6CFLVO</td>
<td>X'120'</td>
<td>X'04'</td>
<td>Input</td>
<td>Old structure CF level (rebuild) or primary structure CF level (duplexing rebuild). For a primary structure CF level, this can be a composite CF level, which is at least as high as a CF level as that which has been previously reported back to any CQS as the primary structure CF level.</td>
</tr>
<tr>
<td>SS6CFLVN</td>
<td>X'124'</td>
<td>X'04'</td>
<td>Input</td>
<td>New structure CF level (rebuild) or secondary structure CF level (duplexing rebuild). For a secondary structure CF level, this can be a composite CF level, which is at least as high as a CF level as that which has been previously reported back to any CQS as the primary structure CF level.</td>
</tr>
<tr>
<td>SS6CFNMS</td>
<td>X'128'</td>
<td>X'04'</td>
<td>Input</td>
<td>CF name in which simplex structure is located (MVS switched to simplex structure).</td>
</tr>
<tr>
<td>SS6VALFL</td>
<td>X'12C'</td>
<td>X'02'</td>
<td>Input</td>
<td>Validity flags (EEPLSSCVALIDITYFLAGS).</td>
</tr>
<tr>
<td></td>
<td>X'12E'</td>
<td>X'02'</td>
<td></td>
<td>Not used.</td>
</tr>
<tr>
<td>SS6DUPST</td>
<td>X'130'</td>
<td>X'08'</td>
<td>Input</td>
<td>Last duplexing rebuild start time (STCK). The last duplexing rebuild for this structure was initiated at this time.</td>
</tr>
<tr>
<td>SS6DUPET</td>
<td>X'138'</td>
<td>X'08'</td>
<td>Input</td>
<td>Last duplexing rebuild end time (STCK). The last duplexing rebuild stopped for this structure occurred at this time.</td>
</tr>
<tr>
<td>SS6UNAVT</td>
<td>X'140'</td>
<td>X'08'</td>
<td>Input</td>
<td>Last structure temporarily unavailable time (STCK). The structure becomes temporarily unavailable because a system-managed rebuild has been initiated, a duplexing rebuild has been initiated, or a duplexing rebuild has stopped.</td>
</tr>
<tr>
<td>SS6AVT</td>
<td>X'148'</td>
<td>X'08'</td>
<td>Input</td>
<td>Last structure available time (STCK). The structure last became available at this time, after initiation of a system-managed rebuild, initiation of a duplexing rebuild, or stopping of a duplexing rebuild.</td>
</tr>
<tr>
<td></td>
<td>X'150'</td>
<td>X'38'</td>
<td></td>
<td>Unused.</td>
</tr>
</tbody>
</table>

Structure Checkpoint Statistics Record for CQS

Structure checkpoint statistics are gathered only by the CQS that is the master of the structure checkpoint process. A CQS has access only to the data it gathers. Each CQS keeps structure checkpoint statistics for the last three checkpoints for which it was the master. Structure checkpoint data is not reset at the end of a structure checkpoint.

Table 15 on page 65 describes the Structure Statistics user exit routine structure checkpoint statistics record.
Table 15. Structure Checkpoint Statistics Record

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS7ID</td>
<td>X‘00’</td>
<td>X‘08’</td>
<td>Input</td>
<td>Eyecatcher CQSSSTT7.</td>
</tr>
<tr>
<td>SS7LN</td>
<td>X‘08’</td>
<td>X‘04’</td>
<td>Input</td>
<td>Length of valid data.</td>
</tr>
<tr>
<td>SS7PVSN</td>
<td>X‘0C’</td>
<td>X‘04’</td>
<td>Input</td>
<td>Parameter List Version Number.</td>
</tr>
<tr>
<td>SS7FLAG1</td>
<td>X‘10’</td>
<td>X‘01’</td>
<td>Input</td>
<td>Flag byte.</td>
</tr>
<tr>
<td></td>
<td>X‘11’</td>
<td>X‘03’</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>SS7ENCNT</td>
<td>X‘14’</td>
<td>X‘04’</td>
<td>Input</td>
<td>Number of structure checkpoint statistics entries in record.</td>
</tr>
<tr>
<td>SS7ENLEN</td>
<td>X‘18’</td>
<td>X‘04’</td>
<td>Input</td>
<td>Length of structure checkpoint statistics entry</td>
</tr>
<tr>
<td>SS7CUR</td>
<td>X‘1C’</td>
<td>X‘04’</td>
<td>Input</td>
<td>Offset to current structure checkpoint statistics entry.</td>
</tr>
<tr>
<td>SS7STATS</td>
<td>X‘20’</td>
<td></td>
<td>Start of structure checkpoint statistics entries. See Table 16 on page 65 for a description of the structure checkpoint statistics entry.</td>
<td></td>
</tr>
</tbody>
</table>

Structure Checkpoint Statistics Entry for CQS

Structure checkpoint statistics are gathered only by the CQS that is the master of the structure checkpoint process. A CQS has access only to the data it gathers. Each CQS keeps structure checkpoint statistics for the last three checkpoints for which it was the master. Structure checkpoint data is not reset at the end of a structure checkpoint.

Table 16 describes the Structure Statistics user exit routine structure checkpoint statistics entry.

Table 16. Structure Checkpoint Statistics Entry

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS7RETCD</td>
<td>X‘00’</td>
<td>X‘08’</td>
<td>Input</td>
<td>Return Code for this Structure Checkpoint</td>
</tr>
<tr>
<td>SS7OQSCB</td>
<td>X‘08’</td>
<td>X‘08’</td>
<td>Input</td>
<td>Structure quiesce start time in STCK format</td>
</tr>
<tr>
<td>SS7OQSCCE</td>
<td>X‘10’</td>
<td>X‘08’</td>
<td>Input</td>
<td>Structure quiesce complete time in STCK format</td>
</tr>
<tr>
<td>SS7DSBP</td>
<td>X‘18’</td>
<td>X‘08’</td>
<td>Input</td>
<td>Start data space/data set capture time in STCK format</td>
</tr>
<tr>
<td>SS7DSEP</td>
<td>X‘20’</td>
<td>X‘08’</td>
<td>Input</td>
<td>End data space capture time in STCK format</td>
</tr>
<tr>
<td>SS7RSMB</td>
<td>X‘28’</td>
<td>X‘08’</td>
<td>Input</td>
<td>Structure resume start time in STCK format</td>
</tr>
<tr>
<td>SS7DSE</td>
<td>X‘30’</td>
<td>X‘08’</td>
<td>Input</td>
<td>End data set capture time in STCK format</td>
</tr>
<tr>
<td>SS7CHKE</td>
<td>X‘38’</td>
<td>X‘08’</td>
<td>Input</td>
<td>Time when all system checkpoints completed in STCK format</td>
</tr>
<tr>
<td>SS7PELA</td>
<td>X‘3C’</td>
<td>X‘04’</td>
<td>Input</td>
<td>Number of allocated elements on primary structure</td>
</tr>
<tr>
<td>SS7PEL</td>
<td>X‘40’</td>
<td>X‘04’</td>
<td>Input</td>
<td>Number of elements in use on primary structure</td>
</tr>
<tr>
<td>SS7OLELA</td>
<td>X‘44’</td>
<td>X‘04’</td>
<td>Input</td>
<td>Number of allocated elements on overflow structure</td>
</tr>
<tr>
<td>SS7PLEA</td>
<td>X‘4C’</td>
<td>X‘04’</td>
<td>Input</td>
<td>Number of allocated list entries on primary structure</td>
</tr>
<tr>
<td>SS7PLEU</td>
<td>X‘50’</td>
<td>X‘04’</td>
<td>Input</td>
<td>Number of list entries in use on primary structure</td>
</tr>
<tr>
<td>SS7OLEA</td>
<td>X‘54’</td>
<td>X‘04’</td>
<td>Input</td>
<td>Number of allocated list entries on overflow structure</td>
</tr>
<tr>
<td>SS7OLEU</td>
<td>X‘58’</td>
<td>X‘04’</td>
<td>Input</td>
<td>Number of list entries in use on overflow structure</td>
</tr>
<tr>
<td>SS7WRTS</td>
<td>X‘5C’</td>
<td>X‘04’</td>
<td>Input</td>
<td>Number of SRDS writes required</td>
</tr>
<tr>
<td>SS7OELU</td>
<td>X‘48’</td>
<td>X‘04’</td>
<td>Input</td>
<td>Number of elements in use on overflow structure</td>
</tr>
</tbody>
</table>
CQS Structure Event User-Supplied Exit Routine

The CQS Structure Event user exit routine is called during CQS processing to notify you of an event related to structure processing. For certain events, it also allows you to gather statistics related to the structure. This exit routine is optional.

The Structure Event user exit routine applies to both resource and queue structures, but not all events are applicable to resource structures. The CQS Structure Event exit routine is driven for the following events:

- **Structure Connection**
  - When structure connect occurs, after CQS connects to a structure, but before rebuild or restart is performed for the structure.
  - At structure disconnect; after CQS disconnects from a structure.

- **Checkpoint**
  - When a system checkpoint begin, end, or failure occurs.
  - When a structure checkpoint begin, end, or failure occurs.

  **Restriction**: The Checkpoint event does not apply to resource structures.

- **Structure Rebuild**
  - When a structure copy (rebuild) begin, end, or failure occurs.
  - When a structure recovery (rebuild) begin, end, or failure occurs.

  **Attention**: The structure failure event for a resource structure (only) means that the structure has failed and a new structure could not be reallocated. No structure recovery is done, because resource structures do not support structure recovery.

- **Structure Overflow**
  - When one or more queues moved to the overflow structure.
  - When one or more queues moved from the overflow structure back to the primary structure. This event also indicates when the structure is no longer in overflow mode.

  **Restriction**: The Structure Overflow event does not apply to resource structures.

- **Structure Status Change**
  - When the structure is available again after a loss.
  - When the structure fails.
  - When CQS loses its connection to the structure.
  - When a resource structure fails and is able to allocate a new resource structure.
  - When the log stream becomes available, making the structure available.

- **Structure Repopulation**
  - When the structure fails and CQS is able to allocate a new resource structure.

  The Structure Repopulation event does not apply to queue structures. The client can repopulate the new resource structure with the resource data.

The exit routine is defined as TYPE = STREVENT in the EXITDEF statement in the BPE user exit PROCLIB member. You can specify one or more exit routines of this type. When this exit routine is invoked, all routines of this type are driven in the order specified by the EXITS= keyword.

**Recommendation**: Write the CQS Structure Event exit routine so that it is reentrant. It is invoked AMODE 31.
Contents of Registers on Entry

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of BPE user exit parameter list (mapped by the BPEUXPL macro). See IMS Version 8 Base Primitive Guide and Reference for more information.</td>
</tr>
<tr>
<td>13</td>
<td>Address of two pre-chained save areas. The first save area can be used by the exit routine to save registers on entry. The second save area can be used by routines that are called from the user exit routine.</td>
</tr>
<tr>
<td>14</td>
<td>Return address.</td>
</tr>
<tr>
<td>15</td>
<td>Entry point of the exit routine.</td>
</tr>
</tbody>
</table>

Contents of Registers on Exit

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
<tr>
<td>0</td>
<td>Always set this to zero.</td>
</tr>
</tbody>
</table>

All other registers must be restored.

Routine Parameter Lists

On entry to the Structure Event exit routine, R1 points to a Standard BPE user exit parameter list. Field UXPL_EXITPLP in this list contains the address of the CQS Structure Event user exit routine parameter list (mapped by the CQSSTREX macro).

Connect Parameter List

Table 17 describes the Structure Event user exit routine connect parameter list.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STXPVSN</td>
<td>X'00'</td>
<td>X'04'</td>
<td>Input</td>
<td>Parameter List Version Number (00000001).</td>
</tr>
<tr>
<td>STXEVENT</td>
<td>X'04'</td>
<td>X'04'</td>
<td>Input</td>
<td>Function code</td>
</tr>
<tr>
<td>STXSCODE</td>
<td>X'08'</td>
<td>X'04'</td>
<td>Input</td>
<td>Event Subcode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Connect Event (STXCONDS).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>Structure connect (STXCONN).</td>
</tr>
<tr>
<td>STXCQSID</td>
<td>X'0C'</td>
<td>X'08'</td>
<td>Input</td>
<td>CQS identifier.</td>
</tr>
<tr>
<td>STXCQSVN</td>
<td>X'14'</td>
<td>X'04'</td>
<td>Input</td>
<td>CQS version number.</td>
</tr>
<tr>
<td>STXSTRNM</td>
<td>X'18'</td>
<td>X'10'</td>
<td>Input</td>
<td>Structure name.</td>
</tr>
<tr>
<td>STXSTRVN</td>
<td>X'28'</td>
<td>X'08'</td>
<td>Input</td>
<td>Structure version number (mapped by the CQSSTREX macro).</td>
</tr>
<tr>
<td>STXDSTT1</td>
<td>X'34'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of structure process statistics record for activity performed by CQS processes on this structure for all clients since restart or the last successful structure checkpoint (mapped by the CQSSSTT1 macro). See Table 9 on page 59 for a description of the structure process statistics. For structure disconnect only.</td>
</tr>
</tbody>
</table>
Structure Event User Exit

Table 17. CQS Structure Event User-Supplied Exit Routine Parameter List: Connect (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STXDSTT2</td>
<td>X'38'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of CQS request statistics record for activity performed for CQS processes on this structure for all clients since restart or the last successful structure checkpoint (mapped by the CQSSSTT2 macro). See Table 10 on page 59 for a description of the CQS request statistics record. For structure disconnect only.</td>
</tr>
<tr>
<td>STXDSTT3</td>
<td>X'3C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of data object statistics record for activity performed on data objects in this structure for all clients since restart or the last successful structure checkpoint (mapped by the CQSSSTT3 macro). See Table 11 on page 60 for a description of the data object statistics record. For structure disconnect only.</td>
</tr>
<tr>
<td>STXDSTT4</td>
<td>X'40'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of queue name statistics record for activity performed on queue names in this structure for all clients since restart or the last successful structure checkpoint (mapped by the CQSSSTT4 macro). See Table 12 on page 61 for a description of the queue name statistics record. For structure disconnect only.</td>
</tr>
<tr>
<td>STXDSTT5</td>
<td>X'44'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of MVS request statistics record for activity performed by CQS processes on this structure for all clients since restart or the last successful structure checkpoint (mapped by the CQSSSTT5 macro). See Table 13 on page 61 for a description of the MVS request statistics record. For structure disconnect only.</td>
</tr>
<tr>
<td>STXDSTT6</td>
<td>X'48'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of rebuild statistics record containing data from the last rebuild in which this CQS acted as master (mapped by the CQSSSTT6 macro). See Table 14 on page 62 for a description of the rebuild statistics record. For structure disconnect only.</td>
</tr>
<tr>
<td>STXDSTT7</td>
<td>X'4C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of structure checkpoint statistics record containing data from the last three structure checkpoints in which this CQS acted as master (mapped by the CQSSSTT7 macro). See Table 15 on page 65 for a description of the structure checkpoint statistics record. For structure disconnect only.</td>
</tr>
</tbody>
</table>

Checkpoint Parameter List

Table 18 describes the Structure Event user exit routine checkpoint parameter list.

Table 18. CQS Structure Event User-Supplied Exit Routine Parameter List: Checkpoint

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STXPVSN</td>
<td>X'00'</td>
<td>X'04'</td>
<td>Input</td>
<td>Parameter List Version Number (00000001).</td>
</tr>
<tr>
<td>STXEVENT</td>
<td>X'04'</td>
<td>X'04'</td>
<td>Input</td>
<td>Structure Event Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>STXSCODE</td>
<td>X'08'</td>
<td>X'04'</td>
<td>Input</td>
<td>Structure Event Subcode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>STXCQSID</td>
<td>X'0C'</td>
<td>X'08'</td>
<td>Input</td>
<td>CQS identifier.</td>
</tr>
<tr>
<td>STXCQSVN</td>
<td>X'14'</td>
<td>X'08'</td>
<td>Input</td>
<td>CQS version number.</td>
</tr>
<tr>
<td>STXSTRNM</td>
<td>X'18'</td>
<td>X'10'</td>
<td>Input</td>
<td>Structure Name.</td>
</tr>
<tr>
<td>STXCMQCS</td>
<td>X'28'</td>
<td>X'08'</td>
<td>Input</td>
<td>CQS identifier of the master CQS performing the checkpoint process. For system checkpoint, this is the same as the CQS identifier.</td>
</tr>
</tbody>
</table>
### Table 18. CQS Structure Event User-Supplied Exit Routine Parameter List: Checkpoint (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STXCFLG1</td>
<td>X'30'</td>
<td>X'01'</td>
<td>Input</td>
<td>Flag byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'80'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This CQS is the master of the process. The CQS identifier and master CQS identifier are the same (STXC1MST).</td>
</tr>
<tr>
<td>N/A</td>
<td>X'31'</td>
<td>X'03'</td>
<td>Input</td>
<td>Reserved.</td>
</tr>
<tr>
<td>STXCSTT1</td>
<td>X'34'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of structure process statistics record for activity performed by CQS processes on this structure for all clients since restart or the last successful structure checkpoint (mapped by the CQSSSTT1 macro). See Table 9 on page 59 for a description of the process statistics record. For system checkpoint end and structure checkpoint end only.</td>
</tr>
<tr>
<td>STXCSTT2</td>
<td>X'38'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of CQS request statistics record for activity performed for CQS requests on this structure for all clients since restart or the last successful structure checkpoint (mapped by the CQSSSTT2 macro). See Table 10 on page 59 for a description of the CQS request statistics record. For system checkpoint end and structure checkpoint end only.</td>
</tr>
<tr>
<td>STXCSTT3</td>
<td>X'3C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of data object statistics record for activity performed on data objects in this structure for all clients since restart or the last successful structure checkpoint (mapped by the CQSSSTT3 macro). See Table 11 on page 60 for a description of the data object statistics record. For system checkpoint end and structure checkpoint end only.</td>
</tr>
<tr>
<td>STXCSTT4</td>
<td>X'40'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of queue name statistics record for activity performed on queue names in this structure for all clients since restart or the last successful structure checkpoint (mapped by the CQSSSTT4 macro). See Table 12 on page 61 for a description of the queue name statistics record. For system checkpoint end and structure checkpoint end only.</td>
</tr>
<tr>
<td>STXCSTT5</td>
<td>X'44'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of MVS request statistics record for activity performed by CQS processes on this structure for all clients since restart or the last successful structure checkpoint (mapped by the CQSSSTT5 macro). See Table 13 on page 61 for a description of the MVS request statistics record. For system checkpoint end and structure checkpoint end only.</td>
</tr>
<tr>
<td>STXCSTT6</td>
<td>X'48'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of rebuild statistics record containing data from the last rebuild in which this CQS acted as master (mapped by the CQSSSTT6 macro). See Table 14 on page 62 for a description of the rebuild statistics record. For system checkpoint end and structure checkpoint end only.</td>
</tr>
<tr>
<td>STXCSTT7</td>
<td>X'4C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of structure checkpoint statistics record containing data from the last three structure checkpoints in which this CQS acted as master (mapped by the CQSSSTT7 macro). See Table 15 on page 65 for a description of the structure checkpoint statistics record. For system checkpoint end and structure checkpoint end only.</td>
</tr>
</tbody>
</table>

### Rebuild Parameter List

*Table 19* describes the Structure Event user exit routine rebuild parameter list.

### Table 19. CQS Structure Event User-Supplied Exit Routine Parameter List: Rebuild

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STXPVSN</td>
<td>X'00'</td>
<td>X'04'</td>
<td>Input</td>
<td>Parameter List Version Number (00000001).</td>
</tr>
<tr>
<td>STXEVENT</td>
<td>X'04'</td>
<td>X'04'</td>
<td>Input</td>
<td>Structure Event Code</td>
</tr>
</tbody>
</table>

3 Structure Rebuild Event (STXRBLD).
Structure Event User Exit

Table 19. CQS Structure Event User-Supplied Exit Routine Parameter List: Rebuild (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STXSCODE</td>
<td>X'08'</td>
<td>X'04'</td>
<td>Input</td>
<td>Structure EventSubcode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 Structure rebuild begin (STXRBLB).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2 Structure rebuild (copy) end (STXCPYE).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3 Structure rebuild (copy) failure (STXCPYF).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4 Structure rebuild failure (STXRBLF).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5 Structure rebuild (recovery) end (STXRCOVE).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6 Structure rebuild (recovery) failure (STXRCOVF).</td>
</tr>
</tbody>
</table>

| STXCQSID   | X'0C'  | X'08'  | Input       | CQS identifier. |
| STXCQSVN   | X'14'  | X'04'  | Input       | CQS version number. |
| STXSTRNM   | X'18'  | X'10'  | Input       | Structure Name. |
| STXRMCQS   | X'28'  | X'08'  | Input       | CQS identifier of the master CQS performing the rebuild process. |
| STXRFLG1   | X'30'  | X'01'  | Input       | Flag byte |
|            |        |        |             | X'80' |

N/A | X'31' | X'03' | Input | Reserved. |

Overflow Parameter List

Table 20 describes the Structure Event user exit routine overflow parameter list.

Table 20. CQS Structure Event User-Supplied Exit Routine Parameter List: Overflow

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STXSCODE</td>
<td>X'08'</td>
<td>X'04'</td>
<td>Input</td>
<td>Structure Event Subcode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 Move queues to overflow. One or more queues were selected as candidates to be moved to the overflow structure and were approved by the Queue Overflow user exit routine (STXTOOFL).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2 Move queues from overflow. One or more queues moved from the overflow structure back to the primary structure, because the queues were drained on the overflow structure. New work for these queues is placed on the primary structure (STXFROFL).</td>
</tr>
</tbody>
</table>

| STXCQSID   | X'0C'  | X'08'  | Input       | CQS identifier. |
| STXCQSVN   | X'14'  | X'04'  | Input       | CQS version number. |
| STXSTRNM   | X'18'  | X'10'  | Input       | Structure Name. |
| STXOMCQS   | X'28'  | X'08'  | Input       | CQS identifier of the master CQS performing the overflow process. |
| STXOFLG1   | X'30'  | X'01'  | Input       | Flag byte |
|            |        |        |             | X'80' |

|            |        |        |             | This CQS is the master of the process. The CQS identifier and master CQS identifier are the same (STX1MST). |
|            |        |        |             | X'40' |

|            |        |        |             | The structure is no longer in overflow mode. This applies only to subcode 2 (STX01END). |
Table 20. CQS Structure Event User-Supplied Exit Routine Parameter List: Overflow (continued)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>X'31'</td>
<td>X'03'</td>
<td>Input</td>
<td>Reserved.</td>
</tr>
<tr>
<td>STXOLSTN</td>
<td>X'34'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of Queue Names entries in the list.</td>
</tr>
<tr>
<td>STXOLSTE</td>
<td>X'38'</td>
<td>X'04'</td>
<td>Input</td>
<td>Length of each Queue Name list entry.</td>
</tr>
<tr>
<td>STXOLSTA</td>
<td>X'3C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Address of Queue Name list. Each Queue Name list entry contains the 16-byte name of a queue that is being moved to or from the overflow structure.</td>
</tr>
</tbody>
</table>

**Status Change Parameter List**

Table 21 describes the Structure Event user exit routine status change parameter list.

Table 21. CQS Structure Event User-Supplied Exit Routine Parameter List: Status Change

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STXPVSN</td>
<td>X'00'</td>
<td>X'04'</td>
<td>Input</td>
<td>Parameter List Version Number (00000003).</td>
</tr>
<tr>
<td>STXEVENT</td>
<td>X'04'</td>
<td>X'04'</td>
<td>Input</td>
<td>Structure Event Code</td>
</tr>
<tr>
<td>STXSCODE</td>
<td>X'08'</td>
<td>X'04'</td>
<td>Input</td>
<td>Structure Event Subcode</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>Structure available again after a loss (STXAVAIL).</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>The structure failed (STXFAIL).</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>CQS lost its connection to the structure (STXLCN).</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>The log stream is becoming available, making the structure available (STXAVLOG).</td>
</tr>
<tr>
<td><strong>Important:</strong></td>
<td></td>
<td></td>
<td></td>
<td>This subcode applies only to queue structures.</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>The log stream is becoming available, making the structure available (STXFLOG).</td>
</tr>
<tr>
<td><strong>Important:</strong></td>
<td></td>
<td></td>
<td></td>
<td>This subcode applies only to queue structures.</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>The structure failed. It needs to be repopulated because this structure does not support structure recovery (STXREPOP).</td>
</tr>
<tr>
<td><strong>Important:</strong></td>
<td></td>
<td></td>
<td></td>
<td>This subcode applies only to resource structures.</td>
</tr>
<tr>
<td>STXCQSID</td>
<td>X'0C'</td>
<td>X'08'</td>
<td>Input</td>
<td>CQS identifier.</td>
</tr>
<tr>
<td>STXCQSVN</td>
<td>X'14'</td>
<td>X'04'</td>
<td>Input</td>
<td>CQS version number.</td>
</tr>
<tr>
<td>STXSTRNM</td>
<td>X'18'</td>
<td>X'10'</td>
<td>Input</td>
<td>Structure Name.</td>
</tr>
<tr>
<td>STXSTYPE</td>
<td>X'28'</td>
<td>X'01'</td>
<td>Input</td>
<td>Input structure type (1 queue structure, 2 resource structures).</td>
</tr>
<tr>
<td>STXRSTVN</td>
<td>X'40'</td>
<td>X'08'</td>
<td>Input</td>
<td>Input structure version.</td>
</tr>
</tbody>
</table>

**CQS Statistics Available through the BPE Statistics User Exit**

You can use the BPE Statistics user exit to gather both BPE and CQS statistics. When the BPE Statistics user exit is driven, field BPESTXP_COMPSTATS_PTR in the BPE Statistics user-supplied exit parameter list, BPESTXP, contains the pointer to the CQS statistics header.
Structure Event User Exit

See the [IMS Version 8: Base Primitive Environment Guide and Reference](#) for detailed information about the BPE Statistics user exit and when it is driven.

Table 22 describes the contents of the CQS Statistics header. The statistics header is mapped by CQSSSTTX.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Field Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>X'08'</td>
<td>Input</td>
<td>Eyecatcher &quot;CQSSTTX&quot;</td>
</tr>
<tr>
<td>X'08'</td>
<td>X'04'</td>
<td>Input</td>
<td>Length of header</td>
</tr>
<tr>
<td>X'0C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Header version number (00000001)</td>
</tr>
<tr>
<td>X'10'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of structures for which statistics are available</td>
</tr>
<tr>
<td>X'14'</td>
<td>X'04'</td>
<td>Input</td>
<td>Number of statistics areas available for each structure</td>
</tr>
<tr>
<td>X'18'</td>
<td>X'04'</td>
<td>Input</td>
<td>Length of all statistics areas for each structure</td>
</tr>
<tr>
<td>X'1C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Offset to statistics area for first structure (offset from CQSSSTTX)</td>
</tr>
<tr>
<td>X'20'</td>
<td>X'04'</td>
<td>Input</td>
<td>CQSSSTAT offset within the statistics area for each structure</td>
</tr>
<tr>
<td>X'24'</td>
<td>X'04'</td>
<td>Input</td>
<td>CQSSSTT1 offset within the statistics area for each structure</td>
</tr>
<tr>
<td>X'28'</td>
<td>X'04'</td>
<td>Input</td>
<td>CQSSSTT2 offset within the statistics area for each structure</td>
</tr>
<tr>
<td>X'2C'</td>
<td>X'04'</td>
<td>Input</td>
<td>CQSSSTT3 offset within the statistics area for each structure</td>
</tr>
<tr>
<td>X'30'</td>
<td>X'04'</td>
<td>Input</td>
<td>CQSSSTT4 offset within the statistics area for each structure</td>
</tr>
<tr>
<td>X'34'</td>
<td>X'04'</td>
<td>Input</td>
<td>CQSSSTT5 offset within the statistics area for each structure</td>
</tr>
<tr>
<td>X'38'</td>
<td>X'04'</td>
<td>Input</td>
<td>CQSSSTT6 offset within the statistics area for each structure</td>
</tr>
<tr>
<td>X'3C'</td>
<td>X'04'</td>
<td>Input</td>
<td>CQSSSTT7 offset within the statistics area for each structure</td>
</tr>
<tr>
<td>X'40'</td>
<td>X'04'</td>
<td>Input</td>
<td>Reserved</td>
</tr>
<tr>
<td>X'44'</td>
<td>X'04'</td>
<td>Input</td>
<td>Reserved</td>
</tr>
<tr>
<td>X'48'</td>
<td>X'04'</td>
<td>Input</td>
<td>Reserved</td>
</tr>
<tr>
<td>X'4C'</td>
<td>X'04'</td>
<td>Input</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
Chapter 5. Writing a CQS Client

If you want to use CQS to manage resource and queues structures for your own product or service, you must write one or more CQS clients. A CQS client uses CQS requests to communicate with CQS. See Chapter 6, “CQS Client Requests,” on page 85 for a complete description of all the CQS requests.

This chapter explains some of the things you must consider when writing a CQS client. The information in this chapter is written for the programmer who will write the client, but a CQS administrator or system programmer should also read this chapter to become aware of some of the issues involved in designing and writing a CQS client.

In this chapter:
- “Introducing CQS Client Requests”
- “Sequence of CQS Requests Issued by a Client for Queue Structure” on page 74
- “Coding CQS Requests” on page 74
- “CQS Clients and Handling Special Events” on page 81

This chapter contains General-Use Programming Interface information.

Introducing CQS Client Requests

Your primary tool for writing a CQS client is the set of client request macros that CQS provides. These requests allow a client to access CQS or the shared queues on coupling facility list structures. The following list summarizes the CQS requests:

- **CQSBRWSE** Retrieves a copy of a data object from a queue
- **CQSCHKPT** Takes a checkpoint of internal tables or of all data objects on a structure
- **CQSCONN** Connects a client to one or more structures
- **CQSDEL** Deletes one or more data objects from a queue
- **CQSDEREG** Deregisters a client from its CQS, terminating communication with it
- **CQSDISC** Disconnects a client from one or more structures
- **CQSIMFRM** Registers client interest in one or more queues, notifying the client when work exists on the queue
- **CQSMOVE** Moves one or more data objects from one queue to another
- **CQSPUT** Places a data object on a queue
- **CQSQUERY** Requests information about a queue or a structure
- **CQSREAD** Retrieves and locks a copy of a data object from a queue
- **CQSRECVR** Recovers data objects that were moved to the cold queue after a client or CQS cold starts
- **CQSSREG** Registers a client with a CQS, establishing communication
- **CQSSSYNC** Resynchronizes in-doubt data between the client and its CQS after a failure
- **CQSSHUT** Shuts down a CQS
- **CQSUNLCK** Unlocks a data object, making it available to any client
Introducing CQS Client Requests

CQSUPD Updates one or more uniquely named resources on a resource structure

**Important:** Some of the requests support either queue or resource structures only. For detailed information on the CQS client requests, see Chapter 6, “CQS Client Requests,” on page 85.

Sequence of CQS Requests Issued by a Client for Queue Structure

A client uses CQS requests to make use of CQS services and resources. There are certain requests the client must issue to request CQS services, and some of the requests must come in a particular sequence; the sequence of CQS requests is shown in Table 23. Other requests can be issued multiple times, in any order, based on the processing requirements of the client.

Table 23. Sequence for CQS Requests

<table>
<thead>
<tr>
<th>Order</th>
<th>Request</th>
<th>Use this request ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CQSREG</td>
<td>To establish communications with CQS.</td>
</tr>
<tr>
<td>2</td>
<td>CQSCONN</td>
<td>To connect to a particular structure.</td>
</tr>
<tr>
<td>3</td>
<td>CQSRSYNC</td>
<td>To resolve indoubt work with CQS.</td>
</tr>
<tr>
<td>4</td>
<td>CQSRECVR†</td>
<td>After a CQS cold start to recover specific data objects.</td>
</tr>
<tr>
<td>5</td>
<td>CQSINFRM</td>
<td>To register interest in specific queue names.</td>
</tr>
<tr>
<td>6</td>
<td>Other CQS requests</td>
<td>To process work. Examples of these other requests are: CQSBRWSE, CQSPUT, and CQSREAD.</td>
</tr>
<tr>
<td>7</td>
<td>CQSDISC</td>
<td>To disconnect from a structure.</td>
</tr>
<tr>
<td>8</td>
<td>CQSSHUT</td>
<td>To request CQS to shutdown. The client could also use CQSDISC ... CQSSHUT=YES to disconnect from a structure and request a CQS shutdown, rather than issuing just the CQSSHUT request.</td>
</tr>
<tr>
<td>9</td>
<td>CQSDEREG</td>
<td>To end communications with CQS.</td>
</tr>
</tbody>
</table>

**Note:**

† A client can issue the CQSRECVR and CQSINFRM requests in any order and at any time following the CQSRSYNC request. The client should, however, issue both of these requests before starting any real work with CQS.

Coding CQS Requests

The usage section for each request (see Chapter 6, “CQS Client Requests,” on page 85) describes the detail for each of the keywords, parameters, and variables for the CQS requests, but there are a few subjects that apply to all of the requests. These global usage considerations are described below, and are not described in each request’s usage section.

Authorization for CQS

CQS provides two interfaces for its clients: the authorized interface and the non-authorized interface. CQS automatically selects and initializes the correct interface environment based on the client’s state when the client issues a CQSREG request. If client is authorized (in supervisor state with PSW key 0 to 7), CQS
initializes the authorized interface environment. If client is not authorized (in problem state with key 8 or greater), CQS initializes the non-authorized interface environment.

Which interface CQS assigns to the client determines the allowed environments for all subsequent CQS requests and all client exit routines driven by CQS. In general, when a client makes a CQS request, its PSW state and key must be the same as they were when it issued the CQSREG request.

Environmental Requirements for CQS

For CQS requests (other than CQSREG and CQSDEREG), the environmental requirements depend on the CQS interface assigned to the client.

Table 24 shows the environment for clients using the authorized CQS interface:

<table>
<thead>
<tr>
<th>Environment</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td>Supervisor state and PSW key 0-7 (PSW key must match the PSW key when the CQSREG request was issued)</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>Any, however, PASN must equal the primary address space in which the CQSREG request was issued</td>
</tr>
<tr>
<td>AMODE</td>
<td>31</td>
</tr>
<tr>
<td>ASC Mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Home address space</td>
<td>Any</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for interrupts</td>
</tr>
<tr>
<td>Control parameters</td>
<td>In primary address space</td>
</tr>
</tbody>
</table>

Table 25 shows the environment for clients using the non-authorized CQS interface:

<table>
<thead>
<tr>
<th>Environment Aspect</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td>Problem state or PSW key 8 (PSW key must match the PSW key when the CQSREG request was issued)</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>None (PASN=SASN=HASN)</td>
</tr>
<tr>
<td>AMODE</td>
<td>31</td>
</tr>
<tr>
<td>ASC Mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Home address space</td>
<td>Address space in which CQSREG was issued</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for interrupts</td>
</tr>
<tr>
<td>Control parameters</td>
<td>In primary address space</td>
</tr>
</tbody>
</table>
The environmental requirements for the CQS register and deregister requests (CQSREG and CQSDEREG) are different from all of the other CQS requests. Authorized clients must issue CQSREG and CQSDEREG requests in the environment shown in Table 26.

Table 26. Environment for CQSREG and CQSDEREG Requests Using the Authorized Interface

<table>
<thead>
<tr>
<th>Environment Aspect</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td>Supervisor state and PSW key 0-7</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>None (PASN=SASN=HASN)</td>
</tr>
<tr>
<td>AMODE</td>
<td>31</td>
</tr>
<tr>
<td>ASC Mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for interrupts</td>
</tr>
<tr>
<td>Control parameters</td>
<td>In primary address space</td>
</tr>
</tbody>
</table>

Non-authorized clients must issue CQSREG and CQSDEREG requests in the environment shown in Table 27.

Table 27. Environment for CQSREG and CQSDEREG Requests Using the Non-Authorized Interface

<table>
<thead>
<tr>
<th>Environment Aspect</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization</td>
<td>Problem state or PSW key 8</td>
</tr>
<tr>
<td>Dispatchable unit mode</td>
<td>Task</td>
</tr>
<tr>
<td>Cross memory mode</td>
<td>None (PASN=SASN=HASN)</td>
</tr>
<tr>
<td>AMODE</td>
<td>31</td>
</tr>
<tr>
<td>ASC Mode</td>
<td>Primary</td>
</tr>
<tr>
<td>Locks</td>
<td>No locks held</td>
</tr>
<tr>
<td>Interrupt status</td>
<td>Enabled for interrupts</td>
</tr>
<tr>
<td>Control parameters</td>
<td>In primary address space</td>
</tr>
</tbody>
</table>

Using Registers with CQS Requests

All CQS requests use registers R0, R1, R14, and R15 as work registers. When a CQS request returns control to the caller, the contents of these registers are not the same as they were before the macro call. R15 contains a return code, and R0 contains a reason code from the CQS interface (see Return Codes and Reason Codes for CQS Requests on page 79). The contents of registers R2 through R13 remain unchanged after a CQS request, except for registers specified as output parameters for the particular request.

All CQS requests require register R13 to point to a standard 72-byte save area. No other registers are required to contain any particular value when a CQS request is issued, except for registers specified as input parameters for the particular request.

Coding Parameters for CQS Requests

For all of the parameters (shown in the syntax diagrams as, for example, parameter) that are not literals, CQS expects either an address or a value. For
example, for the \textit{cqstoken} on a CQSREAD request, CQS expects the address of
the 16-byte CQS token, but for the \textit{buffersize}, CQS expects a 4-byte buffer size.

To pass an address or a parameter value to CQS, you can code the parameter for
the CQS request in one of three ways:

1. Use a register

   To use a register, you must load the address or the parameter value into one of
   the general purpose registers, then use that register (enclosed in parentheses)
   for the parameter in the CQS request.

   \begin{verbatim}
   LA 5,TOKEN
   CQSREAD FUNC=READ,CQSTOKEN=(5),...
   :
   TOKEN DS XL16
   \end{verbatim}

   \textit{Figure 17. Passing an Address for Register}

2. Use a symbol

   To use a symbol name, you must define a symbol that contains the address or
   the parameter value, then use that symbol for the parameter in the CQS request.

   \begin{verbatim}
   L 4,MYBUFSIZE
   CQSREAD FUNC=READ,BUFSE=MYBUFSIZE=(4),...
   :
   MYBUFSIZE DC F'00000024'
   \end{verbatim}

   \textit{Figure 18. Passing a Value for Register}

3. Use a symbol value

   To use a symbol value, you must define a symbol or an equate that contains the
   parameter value, then use that symbol (preceded by the at-sign, @, and
   enclosed in parentheses) for the parameter in the CQS request.

   \begin{verbatim}
   CQSREAD FUNC=READ,BUFSIZE=MYBUFSIZE,...
   :
   MYBUFSIZE DC F'00000024'
   \end{verbatim}

   \textit{Figure 20. Passing a Value for Symbol}
CQSCALL FUNC=READ,CQSTOKEN=@(TOKEN),...

: 
TOKEN DC XL16'0000A76B55CFF00'

Figure 21. Passing a Value for Symbol Value

CQSREAD FUNC=READ,BUFSIZE=@(MYBUFLEN),...

: 
MYBUFLEN EQU 24

Figure 22. Passing an Equate for Symbol Value

Coding Literals for CQS Requests
A number of CQS request macros have parameters that use a literal (for example, the LOCAL parameter on the CQSREAD request macro). A macro invocation can use either combinations of literal parameters or the OPTWORD1 parameter to pass four bytes containing flags that represent the literals. When you use the OPTWORD1 parameter, you obtain the literal equates by using the DSECT function of each request macro. The equates that represent the literal values are added together in a regular storage location.

Requirement: A macro invocation can use either the literal parameters or the OPTWORD1 parameter, not both. When a macro invocation includes the OPTWORD1 parameter, the value passed on this parameter must include one equate for each literal parameter supported by the macro. For example, the CQSREAD request has three literal parameters: LOCAL, PARTIAL, and QPOS. The value you pass on the OPTWORD1 parameter must include one equate for the LOCAL parameter, one equate for the PARTIAL parameter, and one equate for the QPOS parameter.

To code a CQSREAD request using a series of literal parameters, use CQSREAD
FUNC=READ,...,QPOS=FIRST,LOCAL=YES....

To code the same CQSREAD request using the OPTWORD1 parameter, use the example shown in Figure 23:

L R2,=A(CQSREAD_QPOSF+CQSREAD_LCLY+CQSREAD_PRTLY)
CQSREAD FUNC=READ,...,OPTWORD1=(R2),...
.
.
.
CQSREAD FUNC=DSECT GENERATE CQSREAD EQUs

Figure 23. Coding CQSREAD with the OPTWORD1 parameter

Using an ECB with CQS Requests
Some requests allow you to use an MVS event control block (ECB). If you specify an ECB (ECB=ecbaddress), the client immediately receives control after issuing the request, but must at some time be sure to wait for the request to post the ECB. If you do not specify an ECB, CQS does not return control to the client until CQS completes its processing for the request.
Using Lists in the CQS Requests

Some of the CQS requests have a LIST keyword, which specifies the address of a parameter list entry. This keyword specifies the address of the first list entry. If you want to pass multiple list entries, you must ensure that they all reside in contiguous storage, that is, the next entry must begin at the first byte following the current entry. All lists must be contiguous, even if they are not aligned on word or fullword boundaries.

Return Codes and Reason Codes for CQS Requests

With the exception of CQSREG and CQSDEREG, each CQS request returns two sets of return and reason codes. One set is returned by the CQS interface, and indicates the success or failure of sending the request to the CQS address space (these are returned in R15 and R0). The other set is returned by the CQS address space, and reflects the success or failure of the particular CQS request being made (these are returned in the fields indicated by the RETCODE and RSNCODE parameters on the CQS request macro).

When you make a CQS request, the request must travel through the CQS interface from the client address space to the CQS address space. The CQS interface returns information about the success or failure of the sending of the request in registers R15 and R0. After issuing a CQS request macro, have your code check the value in R15 first. If the value in R15 is zero, then the CQS interface successfully sent the request to the CQS address space. If R15 is not zero, the CQS interface was unable to send the request to the CQS address space, and R0 contains a reason code that explains the error.

The return and reason codes from the CQS request itself are returned in the fields specified with the RETCODE and RSNCODE parameters coded on the CQS request macro. The values returned in these fields are valid only if the CQS interface return code (R15) is zero. If the interface return code in R15 is not zero after you issue a CQS request macro, then the values in the RETCODE and RSNCODE fields are not predictable, and you should not use them.

For synchronous requests (that is, requests in which the ECB parameter was not coded), the RETCODE and RSNCODE fields are set after your module receives control back from the request macro, and you can use them immediately. For asynchronous requests (that is, requests in which the ECB parameter was coded), the RETCODE and RSNCODE fields are set only after the ECB is POSTed by CQS. Do not check the RETCODE and RSNCODE fields until you have issued a WAIT on the ECB you specified on the request, and that WAIT has returned.

The CQSREG and CQSDEREG requests are exceptions to the previous description of return and reason codes. CQSREG and CQSDEREG register and deregister a client with the CQS interface, but do not actually send a request across the interface to the CQS address space. CQSREG and CQSDEREG have only a single set of return and reason codes, and these are immediately available upon return from the register or deregister request. The return code is set both in register 15 and in the field specified by RETCODE on the request macro. The reason code is set both in register 0 and in the field specified by RSNCODE on the request macro.

The CQS interface issues the return and reason codes shown in Table 28 on page 80. Any CQS request can receive these return and reason codes. Because the
CQS interface performs more extensive checking for non-authorized clients, some of the following return and reason codes can only be received if the client is a non-authorized client.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000008'</td>
<td>X'00000210'</td>
<td>The cqstoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000214'</td>
<td>The connecttoken is invalid.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000430'</td>
<td>The CQS address space is not available.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000600'</td>
<td>The CQS interface is unable to access internal blocks.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000604'</td>
<td>The client is running in problem state or is using an incorrect PSW key.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000608'</td>
<td>The client passed an invalid function code to the CQS interface.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'0000060C'</td>
<td>The client specified an invalid CQS request type.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000610'</td>
<td>CQS was unable to allocate storage to copy the request parameters.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000614'</td>
<td>The total length of all request parameters passed was less than the sum of all parameter lengths.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000618'</td>
<td>The value passed to the interface for the total length of all parameters was either zero or negative.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'0000061C'</td>
<td>The value passed to the interface for the total parameter count was either zero or negative.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000620'</td>
<td>The length of one of the request's parameters was negative.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000624'</td>
<td>The length passed for the structure-call parameter list was invalid.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000628'</td>
<td>Invalid request function code.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'0000062C'</td>
<td>Invalid request parameter list version number.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000630'</td>
<td>An incorrect number of parameters was passed for the requested function.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000634'</td>
<td>A parameter was passed with an incorrect length.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000638'</td>
<td>A parameter was passed by value instead of by address.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'0000063C'</td>
<td>A parameter was passed by address instead of by value.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000640'</td>
<td>The CQS request abended before being sent to the CQS.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000644'</td>
<td>The CQS request abended while CQS was copying the request parameters. This error is usually caused by the client's passing bad parameter data.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000648'</td>
<td>The interface parameter list version passed by the CQS request macro was not valid. This error is probably caused by a difference in versions between the CQS client and the CQS address space the client is trying to use.</td>
</tr>
</tbody>
</table>
All CQS requests have a DSECT function that you can use to include equate statements in your program for all the return and reason codes for the request.

**Recommendation:** Write a program that specifies FUNC=DSECT for all CQS requests so you can determine symbolic variable names to use for the return and reason code values.

**Assembling a Program with CQS Requests**

The CQS request macros are shipped with IMS and are included in the IMS.ADFSMAC data set. When you assemble a program that includes CQS request macros, you must tell the assembler to look for the macros in this data set. You can also copy the members from the IMS data set to another data set, as necessary.

There are no special requirements for link editing a program that includes CQS requests, but you do have to ensure that the IMS.SDFSRESL data set is concatenated with your JOB or STEPLIB DD statement for the client job.

**Example:** To concatenate the IMS.SDFSRESL data set after your MYPROGS.SDFSRESL data set, code your STEPLIB DD statement as shown in Figure 24:

```
STEPLIB DD DSN=MYPROGS.SDFSRESL,DISP=SHR
       DSN=IMS.SDFSRESL,DISP=SHR
```

*Figure 24. STEPLIB DD Statement to Concatenate IMS.SDFSRESL*

**Attention:** The following coexistence considerations exist for CQS:

- An IMS Version 8 CQS supports only IMS Version 8. Down-level IMSs cannot register with an IMS Version 8 CQS.
- User or vendor-written CQS clients that want to register with an IMS Version 8 CQS must be recompiled with the IMS Version 8 CQS macros.
- A Version 6 IMS and a Version 7 IMS can register with the same Version 7 CQS.
- A Version 7 IMS can register only with a Version 7 CQS.
- A Version 6 CQS, a Version 7 CQS, and a Version 8 CQS can connect to the same coupling facility structure.
- A Version 6 CQS, a Version 7 CQS, and a Version 8 CQS can run on the same central processing complex (CPC).

**CQS Clients and Handling Special Events**

A CQS client must be able either to initiate or to participate in many different types of events. This section describes some of these special events and what the CQS client can or must do about them.

**CQS Cold Start**

When CQS cold starts after connecting to a structure that contains data, CQS looks for unresolved work from CQSMOVE or CQSDEL requests. CQS backs out CQSMOVE requests and completes CQSDEL requests. CQS then performs a system checkpoint, and restart is complete.

CQS does not resolve work that is initiated using a CQSREAD request. As a result, data objects might remain on the queues. The client can issue the CQSSRSYNC request...
Handling Special Events

to have CQS move these data objects to the cold queue and notify the client that they exist. The client can then issue a CQSRECVR request to access these data objects.

**Recommendation:** Complete all work initiated using CQSPUT requests because CQS is not aware of these data objects.

Registering Interest in Queues with CQSINFRM

Use the CQSINFRM request to allow CQS to notify the client when a data object exists on a queue or when the queue becomes nonempty. The client must register interest in a queue before it will be notified of work on that queue.

Working with Objects on the Cold Queue using CQS Requests

CQS places objects on the cold queue when either CQS or the client is cold started while there are objects in active structures. A client can use the CQSBRWSE request to examine objects on the cold queue, and then, using the cold-queue token and UOW returned by this request, the client can use a CQSRECVR request to retrieve or delete objects from the cold queue.

When writing a CQS client, you can use the following request to obtain information about objects on the cold queue, including the qnames, data object count, oldest data object timestamp, and newest data object timestamp:

```
CQSQUERY FUNC=QTYPE,QTYPENM=COLDQ
```

Initiating Checkpoints using CQS Requests

A CQS client can initiate a system checkpoint by issuing a CQSCHKPT FUNC=CHKPTSYS request. See “Using CQS System Checkpoint” on page 39 for more information on system checkpoints.

A CQS client can initiate a structure checkpoint by issuing a CQSCHKPT FUNC=CHKPTSTR request. See “Using CQS Structure Checkpoint” on page 39 for more information on structure checkpoints.

Shutting Down CQS

To shut down CQS, clients can either issue the CQSSHUT request or the CQSDISC request with CQSSHUT=YES specified. In either case, CQS terminates when there are no more structure connections. CQS continues to accept input and output requests so that in-progress work can complete. Structure checkpoints are allowed to be issued. New connections are allowed if the CQSDISC request is issued with CQSSHUT=YES, but they are not allowed if the CQSSHUT request is issued.

**Related Reading:**

- For more information on the CQSDISC request, see “CQSDISC Request” on page 107.
- For more information on the CQSSHUT request, see “CQSSHUT Request” on page 155.

Tuning to Improve CQS Performance

You can improve CQS performance by carefully selecting the parameters you use with the CQSQUERY, CQSDEL, and CQSINFRM requests.
Related Reading: For more information on these tuning recommendations, see “CQSQUERY Request” on page 127, see “CQSDEL Request” on page 102, and see “CQSINFRM Request” on page 112.
Chapter 6. CQS Client Requests

This chapter describes the format, usage, parameters, and return and reason codes of the CQS client requests:

- “CQSBRWSE Request” on page 86
- “CQSCHKPT Request” on page 93
- “CQSCCONN Request” on page 96
- “CQSDEL Request” on page 102
- “CQSDEREG Request” on page 106
- “CQSDISC Request” on page 107
- “CQSINFRM Request” on page 112
- “CQSMOVE Request” on page 116
- “CQSPUT Request” on page 120
- “CQSQUERY Request” on page 127
- “CQSREAD Request” on page 136
- “CQSRECVR Request” on page 141
- “CQSREG Request” on page 146
- “CQSRSYNC Request” on page 148
- “CQSSHUT Request” on page 155
- “CQSUNLCK Request” on page 156
- “CQSUPD Request” on page 161

This chapter contains General-Use Programming Interface information.

Using CQS Client Requests

CQS clients communicate with the CQS address space using a general-use interface consisting of a number of S/390 assembler macros, called CQS requests. Using these requests, CQS clients can communicate with the CQS and manipulate client data on shared coupling facility structures.

Use these requests if you are writing or maintaining a CQS client. You do not need to use them if you are using an IBM-supplied client, such as an IMS control region.

Some CQS requests support wildcard parameters. Wildcard parameters allow you to specify multiple resources whose names match the wildcard parameter mask. The size of a wildcard parameter can be from one character to the maximum number of characters supported for the resource. The alphanumeric name can include one or more specialized characters and an asterisk or percent sign. An asterisk can be replaced by zero, one, or more characters to create a valid resource name. A percent sign can be replaced by exactly one character to create a valid resource name. The wildcard parameter asterisk (*) represents ‘ALL’. However, depending on the installation, other wildcard parameters can mean all. For example, the wildcard parameter %%% means ALL to an installation whose resource names are all 4 characters long.
CQSBRWSE

CQSBRWSE Request

Format for CQSBRWSE

**BROWSE Function of CQSBRWSE:** Use the BROWSE function of a CQSBRWSE request to retrieve a copy of a data object from a specific queue.

```plaintext
CQSBRWSE—FUNC=BROWSE—CQSTOKEN=cqstokenaddress
CONTOKEN=connecttokenaddress—PARAM=parmaddress—BRWTOKEN=browsetokenaddress
QNAME=queueaddress—BUFFER=bufferaddress—BUFSIZE=buffersize
OBJSIZE=dataobjectsizeaddress—UOW=uowaddress
TIMESTAMP=timestampaddress—ECB=ecbaddress
RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress
```

**BRWSOBJ Function of CQSBRWSE:** Use the BRWSOBJ function of a CQSBRWSE request to browse one or more resource data objects of a specified type from a resource structure.

```plaintext
CQSBRWSE—FUNC=BRWSOBJ—CQSTOKEN=cqstokenaddress
CONTOKEN=connecttokenaddress—PARAM=parmaddress—BRWTOKEN=browsetokenaddress
LIST=resourcelistaddress—COUNT=resourcelistcount—LISTVER=1
LISTVER=listversion
BUFFER=bufferaddress—BUFSIZE=buffersize—OBJSIZE=dataobjectsizeaddress
ECB=ecbaddress
RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress
```

**COMPLETE Function of CQSBRWSE:** Use the COMPLETE function of a CQSBRWSE request to indicate to CQS that a CQSBRWSE request associated with a particular browse token is complete.

```plaintext
CQSBRWSE—FUNC=COMPLETE—CQSTOKEN=cqstokenaddress
CONTOKEN=connecttokenaddress—_PARAM=parmaddress—BRWTOKEN=browsetokenaddress
QNAME=queueaddress—BUFFER=bufferaddress—BUFSIZE=buffersize
OBJSIZE=dataobjectsizeaddress—UOW=uowaddress
TIMESTAMP=timestampaddress—ECB=ecbaddress
RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress
```
**CONTINUE Function of CQSBRWSE:** Use the CONTINUE function of a CQSBRWSE request if a previous CQSBRWSE request retrieved partial data and you want to retrieve the rest of the data object.

**DSECT Function of CQSBRWSE:** Use the DSECT function of a CQSBRWSE request to include equate (EQU) statements in your program for the CQSBRWSE parameter list length and CQSBRWSE return and reason codes.

**Usage of CQSBRWSE**

A CQSBRWSE FUNC=BROWSE request retrieves a copy of a data object from a specific queue on a queue structure. The first CQSBRWSE FUNC=BROWSE request takes a snapshot of the data objects meeting the selection criteria and passes back a copy of the first data object. The data object is neither deleted nor locked. It can be accessed by any subsequent CQS request. Each subsequent CQSBRWSE FUNC=BROWSE request retrieves a copy of the next data object. The data object is returned in the client buffer provided on the CQSBRWSE request. The size of the data object is passed to the client.

A browse token maintains the cursor position of the data objects being browsed. A CQSBRWSE FUNC=BROWSE request with a zero browse token passes back the first data object. A CQSBRWSE FUNC=BROWSE request with a non-zero browse token retrieves the next data object on the queue associated with the browse token. If the data object returned is the last data object on the queue, CQS invalidates the browse token and frees any data structures associated with that browse token.

A CQSBRWSE FUNC=BRWSOBJ request retrieves information on one or more data objects from a resource structure. The first CQSBRWSE FUNC=BRWSOBJ request takes a snapshot of the data objects meeting the selection criteria and passes back information on one or more of those data objects. As many data object entries as fit are returned in the client buffer provided on the CQSBRWSE request. Each subsequent CQSBRWSE FUNC=BRWSOBJ request retrieves the next set of data object entries. A browse token maintains the cursor position of the data objects being browsed. A CQSBRWSE FUNC=BRWSOBJ request with a zero browse token retrieves information on as many data objects as fit in the buffer. A CQSBRWSE FUNC=BRWSOBJ request with a non-zero browse token retrieves the next group of data object entries. If the buffer contains information on the last
data object being browsed, CQS invalidates the browse token and frees any data
structures associated with the browse token.

When a CQSBRWSE FUNC=BROWSE request is issued and the buffer passed is
not large enough to hold the next data object, partial data is returned. The buffer is
filled with as much of the data object as can fit. The CQSBRWSE
FUNC=CONTINUE request retrieves the rest of the data object.

Partial data is not returned on a CQSBRWSE FUNC=BRWSOBJ request. The
CQSBRWSE FUNC=CONTINUE request is not supported for a resource structure
because CQSBRWSE FUNC=BRWSOBJ does not return partial data.

A CQSBRWSE FUNC=COMPLETE request indicates to CQS that the CQSBRWSE
request associated with the browse token is complete. The browse token from the
prior CQSBRWSE request is required. CQS invalidates the browse token and frees
any data structures associated with it. The client should issue a CQSBRWSE
FUNC=COMPLETE request if it is not retrieving all of the data objects on the
specified queue.

Attention: The cursor position of a CQSBRWSE FUNC=BROWSE or
CQSBRWSE FUNC=CONTINUE request can be lost due to a CQS restart, a client
restart, structure recovery, structure copy, or the browse table timing out. The
browse table times out after approximately one hour. A CQSBRWSE request is not
recoverable across a CQS or client failure. The client must reissue the CQSBRWSE
request after such a failure. The data object is not locked on a CQSBRWSE
request, so it is possible that one or more of the objects snapped by the first
CQSBRWSE FUNC=BROWSE request are no longer available because of another
CQSREAD, CQSDEL, CQSMOVE request, or overflow threshold processing.
CQSBRWSE FUNC=BROWSE simply skips objects that are no longer available. If
overflow threshold processing occurs after the initial CQSBRWSE FUNC=BROWSE
request and the queue is moved to the overflow structure, any subsequent
CQSBRWSE FUNC=BROWSE request with browse token results in an error that
indicates no objects found. Reissue the CQSBRWSE FUNC=BROWSE request with
a browse token of zeroes, so that CQS can take a snapshot of the queue on the
overflow structure. If the current position is lost because a browse table timed out, a
CQSBRWSE FUNC=CONTINUE request is rejected.

Parameter Description:

**BRWTOKEN=browsetokenaddress**

Input and output parameter that specifies the address of the 16-byte browse
token. The browse token is used to maintain the cursor position of the data
object or objects being browsed.

The browse token should be set to zero on the initial CQSBRWSE request.
The browse token returned by CQS on a CQSBRWSE FUNC=BROWSE or
FUNC=BRWSOBJ request should be passed as input on a subsequent
CQSBRWSE=BROWSE, CONTINUE, COMPLETE, or BRWSOBJ request.

On output, the browse token uniquely identifies the current data object
being browsed, which is returned in the buffer identified by BUFFER.

For a CQSBRWSE FUNC=CONTINUE, a CQSBRWSE
FUNC=COMPLETE, or a subsequent CQSBRWSE FUNC=BROWSE
request, BRWTOKEN is an input parameter that specifies the browse token
returned by CQS on the prior CQSBRWSE FUNC=BROWSE request.
BUFFER=bufferaddress

Four-byte input parameter that specifies the address of a client buffer that
holds information retrieved about one or more data objects.

For CQSBRWSE FUNC=BROWSE, the client buffer contains a copy of the
data object retrieved from the queue on a queue structure.

For CQSBRWSE FUNC=BRWSOBSJS, the client buffer contains the count of
data object entries and one or more data object entries. Each data object
entry contains information about one resource data object retrieved from the
resource structure. The buffer is filled with as many data object entries as
can fit in the buffer. Each data object entry contains information about a
browsed data object such as the resource ID, the completion code,
resource ID status, version, owner, client data1, optional client data2, and
user data that was passed in the input list. If the size of the information is
greater than the buffer size passed by the client, the buffer is filled with as
many resource entries as can fit. The BUFFER is mapped by the
CQSBRWSB DSECT.

The resource ID status indicates how the resource ID in the data object
entry is associated with the input parameter. With this information, you can
tie the input parameter to the data object entries that are generated in the
output buffer. The following are possible resourceid status:

- Specific parameter
  A specific resource ID. This data object entry contains the resource ID
  that matches the input parameter.

- Wildcard parameter
  A wildcard parameter was specified. This data object entry contains the
  wildcard parameter and a completion code. This data object entry does
  not contain information about a specific resource ID. If the completion
  code is zero, one or more wildcard match list entries follow.

- Wildcard match
  A wildcard parameter was specified. This data object contains information
  about one resource ID that matches the input wildcard parameter. All
  wildcard match list entries follow contiguously after a wildcard parameter
  list entry.

The following are possible completion codes:

X'00000000'
Request completed successfully.

X'00000020'
Resourceid
Resourceid is invalid. The name type must be a decimal number
from 1 to 255.

X'00000024'
CQS internal error.

X'00000040'
No resources matching resource ID, resource type, or owner found.

BUFSIZE=buffersize

Four-byte input parameter that specifies the size of the client buffer.

CLDTOKEN=coldqueuetokenaddress

Output parameter that specifies the address of the 16-byte cold-queue
token for the data object, which, along with the UOW, identifies an object on
the cold queue.
CQSBRWSE

You can use the cold-queue token and UOW on a CQSRECVR request to retrieve or delete objects on the cold queue.

CLIENT=clientnameaddress
Four-byte output parameter that specifies the address of an eight-byte field to contain the name of the client that locked the data object with a CQSREAD request. This parameter is valid only when QTYPE=COLD is specified.

CONTOKEN=connecttokenaddress
Input parameter that specifies the address of the 16-byte connect token that uniquely identifies the client’s connection to a particular coupling facility structure managed by this CQS. The connect token is returned by the CQSCONN request.

CQSTOKEN=cqstokenaddress
Input parameter that specifies the address of the 16-byte CQS registration token that uniquely identifies the client’s connection to CQS. The registration token is returned by the CQSREG request.

COUNT=resourceclistcount
Four-byte input parameter that specifies the number of entries in the resource list.

ECB=ecbaddress
Four-byte input parameter that specifies the address of the MVS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise it is processed synchronously.

LIST=resourceclistaddress
Address of a variable size input parameter that specifies a resource list containing one or more entries. Each entry is a separate browse request. The client must initialize some fields in each entry prior to the CQSBRWSE request. Other fields are returned by CQS upon completion of the request.

The CQSBRWSE list entry DSECT maps the list entries and can be used by the client. Multiple list entries must reside in contiguous storage.

Each list entry contains the following:

resourceid
Twelve-byte input field that contains the unique identifier of the resource(s) to be browsed. The resource ID can be a wildcard parameter. The resource ID is unique in the IMSplex. The resource ID consists of a 1-byte name type followed by an 11-byte client-defined name. The name type ensures uniqueness of client-defined names for resources with the same name type. Resources of different resource types may have the same name type. A valid value for the name type is a decimal number from 1 to 255. The client-defined name has meaning to the client and consists of alphanumeric characters. If you use a wildcard parameter to specify the resource ID, you should also specify the resource type to enhance performance. You must specify the resource ID, resource type, or both.

resourcetype
One-byte input field that specifies the resource type. The resource type is a client-defined physical grouping of resources on the resource structure. Valid values for the resource type are decimal numbers from 1 to 255. If the resource type is greater than the maximum number of
resource types defined by CQS (11), it is folded into one of the existing
resource types. You must specify the resource type, resource ID, or
both.

reserved
Three-byte reserved field.

owner
Eight-byte input parameter that identifies the owner of the resource data
objects to be browsed. The CQSBRWSE request returns only those
resource data objects that are owned by the specific owner. Owner is
an optional parameter.

options
Four-byte input parameter that specifies browse options. Possible
options are:

X'80000000'
Return data2 for the browsed data object(s).

userdata
Four-byte input parameter that specifies user data. This user
data is passed on output for each data object that matches the
input resource ID parameter.

LISTVER=1 listversion
Input parameter that specifies an equate for the list version. The default
value is 1. Use the DSECT function of a CQSBRWSE request to include
equate (EQU) statements in your program for the CQSBRWSE list versions.

OBJSIZE=dataobjectsizetypenumber
Output parameter that specifies the address of a 4-byte area to hold the
size of a data object or data object entry.

If a CQSBRWSE FUNC=Browse request is issued and the size of the
data object is greater than the buffer size passed by the client, the buffer is
filled with as much of the data object as fits. The request receives a return
and reason code indicating partial data returned. The size of the data object
is returned in the location specified by the OBJSIZE parameter. If the size
of the data object is less than or equal to the size of the buffer, the data
object is moved into the buffer and the remainder of the buffer is not
changed.

If a CQSBRWSE FUNC=BRWSOBJ request is issued, as many data
object entries as can fit are moved into the buffer. The client must then
issue a subsequent CQSBRWSE FUNC=BRWSOBJ request to retrieve
the next data object entries. If the buffer is not large enough to hold the
next data object entry, the request receives a return and reason code
indicating the buffer is too small. The size of the next data object entry to
be returned is saved in the location specified by the OBJSIZE parameter.

PARAM=paramaddress
Four-byte input parameter that specifies the address of a parameter list
used by the request to pass parameters to CQS. The length of the storage
area must be at least equal to the EQU value CQSBRWSE_PARM_LEN
(defined using the FUNC=DSECT request).

QNAME=queuenametypenumber
Four-byte output address parameter that specifies the address of a 16-byte queue
name field.
For a CQSBRWSE request that specifies QTYPE=COLD and CLDTOKEN, the queue name field is an output field to contain the original client queue name for the data object being returned. This client queue name contained the data object before it was moved to the cold queue.

For all other CQSBRWSE requests, the queue name field is an input field that specifies the queue name from which the data object is retrieved for all CQSBRWSE requests.

**QTYPE=COLD**
Input parameter that specifies the queue type from which the data object is to be retrieved.

**COLD** Indicates the data object is to be retrieved from the cold queue.

**RETCODE=returncodeaddress**
Output parameter that specifies the address of a 4-byte field to contain the CQSBRWSE return code.

If the return code in register 15 is nonzero, the values in the return and reason code fields are invalid, because the CQS interface detected an error and was unable to send the request to CQS.

**RSNCODE=reasoncodeaddress**
Output parameter that specifies the address of a 4-byte field to contain the CQSBRWSE reason code.

**TIMESTAMP=timestampaddress**
Four-byte output parameter that specifies the address of an eight-byte field to contain the timestamp of when the data object was placed on the queues.

**UOW=uowaddress**
Output parameter that specifies the address of a 32-byte area to hold the unit of work (UOW) of the data object retrieved from the queue. The UOW is a unique identifier generated by the client that stored the data object on the queue (CQSPUT request).

### Return and Reason Codes for CQSBRWSE

Table 29 lists the return and reason code combinations that can be returned for CQSBRWSE requests. Use a CQSBRWSE FUNC=DSECT request to include equate statements in your program for the return and reason codes.

**Table 29. CQSBRWSE Return and Reason Codes**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000000'</td>
<td>X'00000000'</td>
<td>Request completed successfully.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000120'</td>
<td>The buffer size (buffersize) is less than the data object size (dataobjectsize). Partial data is returned.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000124'</td>
<td>The buffer size (buffersize) is too small to contain the next resource data object entry. No partial data is returned.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000128'</td>
<td>No data object to retrieve on queue name (queueName) specified.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'0000012C'</td>
<td>No partial data to return.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000138'</td>
<td>Request complete and the last data object is returned.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'0000013C'</td>
<td>No more data objects to return.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000210'</td>
<td>cqstoken is invalid.</td>
</tr>
</tbody>
</table>
### Table 29. CQSBRWSE Return and Reason Codes (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000008'</td>
<td>X'00000214'</td>
<td>connecttoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000218'</td>
<td>FUNC is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'0000021C'</td>
<td>browsetoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000220'</td>
<td>queuename is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000224'</td>
<td>buffer is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000228'</td>
<td>buffersize is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'0000022C'</td>
<td>dataobjectsize is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000230'</td>
<td>uow is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000234'</td>
<td>browsetoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000250'</td>
<td>Count is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000254'</td>
<td>List address is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'0000027C'</td>
<td>CQSBRWSE FUNC=BROWSE is not allowed for a resource structure. No partial data is returned from a resource structure.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000280'</td>
<td>CQSBRWSE FUNC=BRWSOBS is not allowed for a queue structure.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000284'</td>
<td>Parmlist version is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000288'</td>
<td>List version is invalid.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000400'</td>
<td>A CQSSYNC is required for this structure.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000420'</td>
<td>Structure is inaccessible. Retry request later.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000428'</td>
<td>Current position lost, reissue CQSBRWSE request.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000500'</td>
<td>CQS internal error.</td>
</tr>
</tbody>
</table>

### CQSCHKPT Request

**Format for CQSCHKPT**

**CHKPTSTR Function of CQSCHKPT:** Use the CHKPTSTR function of a CQSCHKPT request to initiate a CQS structure checkpoint for a queue structure. Structure checkpoint is not supported for a resource structure.
CQSCHKPT

**CHKPTSYS Function of CQSCHKPT:** Use the CHKPTSYS function of a CQSCHKPT request to initiate a CQS system checkpoint.

```
CQSCHKPT FUNC=CHKPTSYS CQSTOKEN=cqstokenaddress PARM=parmaddress
```

```
COUNT=count LIST=listaddress RETCODE=returncodeaddress
```

```
RSNCODE=reasoncodeaddress CQSCHKPT FUNC=CHKPTSYS CQSTOKEN=cqstokenaddress
```

```
PARM=parmaddress COUNT=count LIST=listaddress ECB=ecbaddress
```

```
RETCODE=returncodeaddress RSNCODE=reasoncodeaddress
```

```
LISTVER=1
```

```
LISTVER=listversion
```

**DSECT Function of CQSCHKPT:** Use the DSECT function of a CQSCHKPT request to include equate (EQU) statements in your program for the CQSCHKPT parameter list length and CQSCHKPT return and reason codes.

```
CQSCHKPT FUNC=DSECT
```

**Usage of CQSCHKPT**

A CQS client can use a CQSCHKPT request to initiate either a CQS system checkpoint or a structure checkpoint.
For a structure checkpoint, CQS dumps the queues to DASD for each structure specified in the checkpoint list. If the structure is currently in overflow mode, the overflow structure is also dumped to DASD.

For a system checkpoint, CQS logs the internal tables for each structure specified in the checkpoint list. If the structure is currently in overflow mode, CQS also logs the internal tables for the overflow structure.

Parameter Description:

**COUNT**=*count*

Four-byte input parameter that specifies the number of entries in the checkpoint list.

**CQSTOKEN**=*cqstokenaddress*

Input parameter that specifies the address of the 16-byte CQS registration token that uniquely identifies the client’s connection to CQS. The registration token is returned by the CQSREG request.

**ECB**=*ecbaddress*

Four-byte input parameter that specifies the address of the MVS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise it is processed synchronously.

**LIST**=*listaddress*

Four-byte input parameter that specifies the address of the checkpoint list. The checkpoint list should contain an entry for each of the structures for which the client requests a checkpoint.

The CQSCHKPL list entry DSECT maps the list entries and can be used by the client. Multiple list entries must reside in contiguous storage.

Each list entry contains the following:

**connecttoken**

Sixteen-byte input parameter that specifies the connect token returned by the CQSCONN request. The connect token uniquely identifies the client’s connection to a particular coupling facility structure managed by this CQS. This parameter is required.

**compcode**

Four-byte output field to receive the completion code from the request. Possible completion codes are:

- **X'00000000'**: Completed successfully.
- **X'00000004'**: Connect token is invalid.
- **X'00000008'**: CQS checkpoint request not allowed until CQS restart has successfully completed a system checkpoint.
- **X'0000000C'**: A CQSRSYNC is required for this structure.
- **X'00000010'**: Checkpoint already in progress for structure.
- **X'00000014'**: Structure is inaccessible. Retry request later.
- **X'00000018'**: CQS internal error.
- **X'00000020'**: CQSCHKPT FUNC=CHKPTSTR is invalid for a resource structure.
LISTVER=1 | listversion
Input parameter that specifies an equate for the list version. Use the DSECT function of a CQSCHKPT request to include equate (EQU) statements in your program for the CQSCHKPT list versions.

PARM=parmaddress
Four-byte input parameter that specifies the address of a parameter list used by the request to pass parameters to CQS. The length of the storage area must be at least equal to the EQU value CQSCHKPT_PARM_LEN (defined using the FUNC=DSECT request).

RETCODE=returncodeaddress
Output parameter that specifies the address of a 4-byte field to contain the CQSCHKPT return code.
If the return code in register 15 is nonzero, the values in the return and reason code fields are invalid, because the CQS interface detected an error and was unable to send the request to CQS.

RSNCODE=reasoncodeaddress
Output parameter that specifies the address of a 4-byte field to contain the CQSCHKPT reason code.

Return and Reason Codes for CQSCHKPT
Table 30 lists the return and reason code combinations that can be returned for CQSCHKPT requests. Use a CQSCHKPT FUNC=DSECT request to include equate statements in your program for the return and reason codes.

Table 30. CQSCHKPT Return and Reason Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000000'</td>
<td>X'00000000'</td>
<td>Request completed successfully.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000210'</td>
<td>cqstoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000218'</td>
<td>FUNC is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000250'</td>
<td>count is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000254'</td>
<td>listaddress is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000284'</td>
<td>Parmlist version is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000288'</td>
<td>List version is invalid.</td>
</tr>
<tr>
<td>X'0000000C'</td>
<td>X'00000300'</td>
<td>Request succeeded for at least one, but not all, list entries. See compcode for individual errors.</td>
</tr>
<tr>
<td>X'0000000C'</td>
<td>X'00000304'</td>
<td>Request failed for all list entries. See compcode for individual errors.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'0000040C'</td>
<td>CQS shutdown is pending. Client-initiated checkpoint requests are not allowed.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000430'</td>
<td>No CQS address space.</td>
</tr>
</tbody>
</table>

CQSCONN Request

Format for CQSCONN

CONNECT Function of CQSCONN: Use the CONNECT function of a CQSCONN request to connect to one or more coupling facility structures. The coupling facility structures can be queue structures or resource structures.
DSECT Function of CQSCONN: Use the DSECT function of a CQSCONN request to include equate (EQU) statements in your program for the CQSCONN parameter list length and CQSCONN return and reason codes.

Usage of CQSCONN

The CQSCONN request connects a client to one or more coupling facility structures. The client specifies a connect list containing one or more list entries, for which each entry is a separate connect request. If the connection to a structure is successful, a connect token is returned to the client, representing the connection to the structure. The client must specify this token on all subsequent CQS requests for that structure. A maximum of 32 clients can use a CQS address space to connect to a coupling facility structure.

Restriction: The CQSCONN request is not logged for resource structures and does not support the FCCQSSSN keyword. The CQSCONN request does not support the following connect list parameters for a resource structure:

- structureattributes
- overflowstructurename
- structureinformexit
- structureinformparm
- qtypecnt
- qtypelist

A CQSCONN FUNC=CONNECT request must be issued after a CQSREG FUNC=REGISTER request and before any other CQS requests. Also, after a CQS abnormal termination and restart, and after the client has reregistered with CQS, a CQSCONN FUNC=CONNECT request is required before the client can issue any other CQS requests.

Parameter Description:

**COUNT=count**

Four-byte input parameter that specifies the number of list entries in the connect list.

**CQSTOKEN=cqstokenaddress**

Input parameter that specifies the address of the 16-byte CQS registration token.

**LIST=address**

Input parameter that specifies the address of the 16-byte CQS registration token.

**ECB=ecbaddress**

Input parameter that specifies the address of the 8-byte ECB.

**RETCODE=returncodeaddress**

Input parameter that specifies the address of the 4-byte return code.

**RSNCODE=reasoncodeaddress**

Input parameter that specifies the address of the 4-byte reason code.

**LISTVER=listversion**

Input parameter that specifies the version of the list.
token that uniquely identifies the client's connection to CQS. The registration token is returned by the CQSREG request.

**ECB=ecbaddress**

Four-byte input parameter that specifies the address of the MVS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise it is processed synchronously.

**FCCQSSSN=fccqssnaddress**

Four-byte input parameter that specifies the address of the failed client CQS subsystem. When one client takes over for another client, this is the SSN of the CQS that was connected to the failed client.

This keyword is not applicable to a resource structure.

**LIST=listaddress**

Four-byte input parameter that specifies the address of a connect list containing one or more entries. Each entry is a separate request to connect a client to a coupling facility structure. Some fields for each entry must be initialized by the client prior to the CQSCONN request. Other fields are returned by CQS upon completion of the CQSCONN request.

The CQSCONNL list entry DSECT maps the list entries and can be used by the client. Multiple list entries must reside in contiguous storage.

Each list entry contains the following:

**compcode**

Four-byte output field to receive the completion code from the request. Possible completion codes are:

- **X'00000000'** Client connection successful. A connect token is returned to the client.
- **X'00000004'** The client is already connected to the structure through this CQS. A connect token is returned to the client.
- **X'00000008'** structurename is invalid.
- **X'0000000C'** The Structure Event exit routine address was not specified.
- **X'00000010'** The client is already connected to the structure through another CQS. A client can only be connected to a given structure through one CQS. The client is not connected to the structure through this CQS. This does not affect the status of a client connection with another CQS.
- **X'00000014'** CQS internal error.
- **X'00000018'** The client specified the FCCQSSSN= parameter to connect to the structure to take over work for a failed client. CQS could not find a valid system-checkpoint log token for the CQS that was connected to the failed client. CQS issued message CQS0033A, to which the operator replied REJECT.
- **X'0000001C'** The user ID of the client address space is not authorized to connect to the structure.
structureinformexit was specified but is not allowed for a resource structure.

structureinformparm was specified but is not allowed for a resource structure.

structureattributes was specified but is not allowed for a resource structure.

Qtype was specified but is not allowed for a resource structure.

FCCQSSSN was specified but is not allowed for a resource structure.

**structureattributes**

Four-byte input and output parameter field that contains the structure attributes.

+0  Flag byte 1, with the following bits defined:

- X'80' Indicates the specification of the structure "wait for rebuild" attribute. The first client in the sysplex to connect to a structure defines this attribute for all clients. It is returned on the connect request to allow clients to verify that the attribute is set correctly for their needs because it might have been set by a prior client connection.

  The value specified for `structureattributes` remains in effect for the life of the structure, and cannot be changed.

  When set to 0, indicates that client requests to write and retrieve data objects from the structure do not wait for a rebuild to complete.

  When set to 1, indicates that client requests to write and retrieve data objects from the structure must wait for a rebuild to complete.

  The remaining bits in this byte are not used, and must be set to zero.

+1  The next three bytes are not used, and must be set to zero.

**structuretype**

One-byte output parameter field that specifies the structure type as either a queue structure or a resource structure.

**structureversion**

Eight-byte output parameter field that specifies the structure version of the structure to which the client just connected.

**structurename**

Sixteen-byte input parameter field that contains the name of the structure to which the client wants to connect. This parameter is required.

**overflowstructurename**

Sixteen-byte output parameter field to receive the name of the
overflow structure, if one was defined to CQS in the CQS Global Structure Definition PROCLIB member, CQSSGxxx.

This parameter is not applicable to a resource structure.

**connecttoken**

Sixteen-byte output parameter field to receive the connect token that uniquely identifies the client's connection to a particular coupling facility structure managed by this CQS.

**structureeventexit**

Four-byte input parameter field that contains the Structure Event exit routine address. This parameter is required.

**structureeventparm**

Four-byte input parameter field that contains client data that CQS passes to the Structure Event exit routine every time the exit is called. This parameter is optional; set it to zero if you do not want to pass any data to the exit routine.

**structureinformexit**

Four-byte input parameter field that contains the Structure Inform exit routine address. This parameter is optional; set it to zero if you do not have a Structure Inform exit routine.

This parameter is not applicable to a resource structure.

**structureinformparm**

Four-byte input parameter field that contains client data that CQS passes to the Structure Inform exit routine every time the exit is called. This parameter is optional; set it to zero if you do not want to pass any data to the exit routine.

This parameter is not applicable to a resource structure.

**qtypecnt**

Four-byte input parameter field that contains the number of queue type entries in the queue type list. This parameter is optional; set it to zero if you do not have any entries in the queue type list.

This parameter is not applicable to a resource structure.

**qtypelst**

Variable length input area for the queue type list.

This parameter is not applicable to a resource structure.

The length of this area is equal to the value specified for qtypecnt. Each queue type entry is a 1-byte value of a queue type that should not be moved to the overflow structure if the primary structure goes into overflow mode. This parameter is optional.

After a queue type is defined, it remains in effect for the life of the structure, and is not moved to the overflow structure.

If no queue types are listed, the default is for all queue types to be eligible for overflow. This list should only be included if there are certain queue types the client knows should not be moved (perhaps based on the client's use of the queue types).

**Recommendation:** Clients should exclude from processing those queue types that allow multiple objects with the same queue name.
and UOW. CQS cannot recover multiple objects with the same queue name and UOW that are allowed to be moved to the overflow structure.

LISTSIZE=\texttt{listsize}

Four-byte input parameter that specifies the size of the connect list. \texttt{listsize} specifies the total length of all entries in the list, not the length of a single entry.

LISTVER=1 | \texttt{listversion}

Input parameter that specifies an equate for the list version. Use the DSECT function of a CQSCONN request to include equate (\texttt{EQU}) statements in your program for the CQSCONN list versions.

PARM=\texttt{parmaddress}

Four-byte input parameter that specifies the address of a parameter list used by the request to pass parameters to CQS. The length of the storage area must be at least equal to the \texttt{EQU} value CQSCONN\_PARM\_LEN (defined using the FUNC\=DSECT request).

RETCODE=\texttt{returncodeaddress}

Output parameter that specifies the address of a 4-byte field to contain the CQSCONN return code. If the return code in register 15 is nonzero, the values in the return and reason code fields are invalid, because the CQS interface detected an error and was unable to send the request to CQS.

RSNCODE=\texttt{reasoncodeaddress}

Output parameter that specifies the address of a 4-byte field to contain the CQSCONN reason code.

### Return and Reason Codes for CQSCONN

Table \ref{table:cqsc_conn_codes} lists the return and reason code combinations that can be returned for CQSCONN requests. Use a CQSCONN FUNC\=DSECT request to include equate statements in your program for the return and reason codes.

\begin{table}[h]
\centering
\caption{CQSCONN Return and Reason Codes}
\begin{tabular}{|c|c|l|}
\hline
\textbf{Return Code} & \textbf{Reason Code} & \textbf{Meaning} \\
\hline
00000000' & 00000000' & Request completed successfully. \\
\hline
00000004' & 00000100' & The client was previously connected to one or more of the specified structures through this CQS. Client is connected to all structures. \\
\hline
00000008' & 00000100' & \texttt{cqstoken} is invalid. \\
\hline
00000008' & 00000218' & FUNC is invalid. \\
\hline
00000008' & 00000250' & \texttt{count} is invalid. \\
\hline
00000008' & 00000254' & \texttt{listaddress} is invalid. \\
\hline
00000008' & 00000258' & \texttt{listsize} is invalid. \\
\hline
00000008' & 00000284' & Parmlist version is invalid. \\
\hline
00000008' & 00000288' & List version is invalid. \\
\hline
0000000C' & 00000300' & Request succeeded for one but not all list entries. See \texttt{compcode} for individual errors. \\
\hline
0000000C' & 00000304' & Request failed for all list entries. See \texttt{compcode} for individual errors. \\
\hline
\end{tabular}
\end{table}
Table 31. CQSCONN Return and Reason Codes (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000010'</td>
<td>X'0000040C'</td>
<td>CQS shutdown in progress (CQSSHUT). CQS is waiting for all clients to disconnect, and no new client connections are allowed.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000410'</td>
<td>The maximum number of clients are connected to this CQS. This request would exceed the client connection limit. No further client connections are allowed.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000430'</td>
<td>No CQS address space.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000500'</td>
<td>CQS internal error.</td>
</tr>
</tbody>
</table>

### CQSDEL Request

#### Format for CQSDEL

**DELETE Function of CQSDEL:** Use the DELETE function of a CQSDEL request to delete one or more data objects from a queue structure or a resource structure.

```
CQSDELL—FUNC=DELETE—CQSTOKEN=cqstokenaddress
```

```
—CONTOKEN=connecttokenaddress—PARM=parmaddress—COUNT=count
```

```
—LIST=listaddress—ECB=ecbaddress—RETCODE=returncodeaddress
```

```
—RNCODE=reasoncodeaddress—LISTVER=listversion
```

**DSECT Function of CQSDEL:** Use the DSECT function of a CQSDEL request to include equate (EQU) statements in your program for the CQSDEL parameter list length and CQSDEL return and reason codes.

```
CQSDELL—FUNC=DSECT
```

### Usage of CQSDEL

A CQSDEL request deletes one or more data objects from a queue structure or a resource structure. The client specifies a delete list containing one or more list entries, for which each list entry is a separate delete request (either by lock token, by queue name, by queue name and UOW, by resource ID, or by resource type and owner). Each list entry is processed separately and receives its own completion code.

### Parameter Description:

**CONTOKEN=connecttokenaddress**

Input parameter that specifies the address of the 16-byte connect token that uniquely identifies the client's connection to a particular coupling facility structure managed by this CQS. The connect token is returned by the CQSCONN request.
**CQSDEL**

**COUNT=** `count`
Four-byte input parameter that specifies the number of list entries in the delete list.

**CQSTOKEN=** `cqstokenaddress`
Input parameter that specifies the address of the 16-byte CQS registration token that uniquely identifies the client’s connection to CQS. The registration token is returned by the CQSREG request.

**ECB=** `ecbaddress`
Four-byte input parameter that specifies the address of the MVS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise, it is processed synchronously.

**LIST=** `listaddress`
Four-byte input parameter that specifies the address of a delete list containing one or more entries. Each entry is a separate delete request. Some fields in each entry must be initialized by the client prior to the CQSDEL request. Other fields are returned by CQS upon completion of the request.

The CQSDELL list entry DSECT maps the list entries and can be used by the client. Multiple list entries must reside in contiguous storage.

Each list entry contains the following:

**deletetype**
One-byte input parameter field that contains the delete type. This is a required parameter. `deletetype` can be one of the following:

1. Delete by lock token.
2. Delete by queue name.
3. Delete by queue name and unit of work.
4. Delete by resource ID and version.
5. Delete by resource type with the specified owner.

**Recommendation:** For better performance, use delete type 1 or delete type 2 because they are more efficient than delete type 3.

**deleteqpos**
One-byte input parameter field that specifies either that all data objects are to be deleted or the position on the queue of data objects to be deleted. This parameter is only used for delete type 2. `deleteqpos` can be one of the following:

1. Delete all data objects on the queue.
2. Delete the first data object on the queue.
3. Delete the last data object on the queue.

The `locktoken`, `deleteqpos`, and `uow` fields are mutually exclusive.

**reserved**
Two-byte reserved field.

**objdelcnt**
Four-byte output parameter field to receive the number of data objects deleted.
compcode

Four-byte output field to receive the completion code from the request. Possible completion codes are:

- X'00000000' Request completed successfully.
- X'00000004' Invalid deleteqpos (Delete type 2).
- X'00000008' Invalid deletetype.
- X'0000000C' Invalid locktoken (Delete type 1).
- X'00000010' Invalid queuename (Delete type 2 or type 3).
- X'00000014' Invalid uow (Delete type 3).
- X'0000001C' Structure is inaccessible. Retry request later.
- X'00000020' CQS internal error.
- X'00000024' Data object not found on queue (Delete type 2) or on queuename for UOW (Delete type 3), or on resource structure (Delete type 4). It is up to the client to determine whether this case should be treated as an error or not.
- X'00000028' Delete type 1, 2, or 3 is invalid for a resource structure.
- X'0000002C' Delete type 4 or 5 is invalid for a queue structure.
- X'00000030' Resourceid is invalid. The name type must be a decimal number from 1 to 255.
- X'00000034' Version does not match that of an existing resource.
- X'00000038' Resourcetype is invalid. The resource type must be a decimal number from 1 to 255.
- X'0000003C' Owner is invalid. The owner is required for delete type 5.
- X'00000040' Version is invalid. The version must be a number greater than zero.

locktoken

Sixteen-byte input parameter field that contains the lock token. The lock token is returned by the CQSREAD request. This parameter is only used for delete type 1.

The locktoken, deleteqpos, and uow fields are mutually exclusive. The locktoken and queuename fields are also mutually exclusive.

queuename

Sixteen-byte input parameter field that contains the queue name. This parameter is only used for delete types 2 and 3.

The locktoken and queuename fields are mutually exclusive.

uow

Thirty-two-byte input parameter that contains the unit of work. This parameter is only used for delete type 3.

The locktoken, deleteqpos, and uow fields are mutually exclusive.

resourceid

Twelve-byte input parameter that contains the unique identifier of
the resource data object to delete. This parameter is required for
delete type 4. The resourceid, locktoken, queuename, and
resourceype fields are mutually exclusive.

version
Eight-byte input and output parameter that contains the version of
the resource to be deleted. The version specified must match the
version of the resource for the delete request to succeed. The
version is a count of the number of times the resource has been
updated. This parameter is required for delete type 4. If the delete
fails because of version mismatch, the version is returned as
output.

resourcetype
One-byte input parameter that contains the resource type. The
resource type is a client-defined physical grouping of resources on
the resource structure. Valid values for the resource type are
decimal numbers from 1 to 255. If the resource type is greater than
the maximum number of resource types defined by CQS (11), it is
folded into one of the existing resource types. This parameter is
required for delete types 4 and 5. Specify zero to delete all
resources of a resource type that are not owned.

reserved
Three-byte reserved field.

owner Eight-byte input parameter that specifies the owner for which to
delete resources of the specified resource type. This parameter is
required for delete type 5.

LISTVER=1
listversion
Input parameter that specifies an equate for the list version. Use the
DSECT function of a CQSDEL request to include equate (EQU) statements
in your program for the CQSDEL list versions.

PARM=parmaddress
Four-byte input parameter that specifies the address of a parameter list
used by the request to pass parameters to CQS. The length of the storage
area must be at least equal to the EQU value CQSDEL_PARM_LEN (defined
using the FUNC=DSECT request).

RETCODE=returncodeaddress
Output parameter that specifies the address of a 4-byte field to contain the
CQSDEL return code.
If the return code in register 15 is nonzero, the values in the return and
reason code fields are invalid, because the CQS interface detected an error
and was unable to send the request to CQS.

RSNCODE=reasoncodeaddress
Output parameter that specifies the address of a 4-byte field to contain the
CQSDEL reason code.

Return and Reason Codes for CQSDEL
Table 32 on page 106 lists the return and reason code combinations that can be
returned for CQSDEL requests. Use a CQSDEL FUNC=DSECT request to include
equate statements in your program for the return and reason codes.
### CQSDEREG Request

#### Format for CQSDEREG

**DEREGISTER Function of CQSDEREG:** Use the DEREGISTER function of a CQSDEREG request to deregister a client from CQS and invalidate the CQS token.

```
CQSDEREG—FUNC=DEREGISTER—CQSTOKEN=cqstokenaddress—PARM=parmaddress
```

**DSECT Function of CQSDEREG:** Use the DSECT function of a CQSDEREG request to include equate (EQU) statements in your program for the CQSDEREG parameter list length and CQSDEREG return and reason codes.

```
CQSDEREG—FUNC=DSECT
```

#### Usage of CQSDEREG

The CQSDEREG request deregisters a client from CQS and invalidates the CQSTOKEN. Prior to issuing this request, the client should issue the CQSDISC request to disconnect from all structures to which the client has a connection. When this request is successfully completed, no subsequent requests can be made to CQS until a CQSREG request has been made to get a new CQSTOKEN.

**Parameter Description:**

- **CQSTOKEN=cqstokenaddress**
  
  Input parameter that specifies the address of the 16-byte CQS registration token that uniquely identifies the client’s connection to CQS. The registration token is returned by the CQSREG request.
PARM=parmaddress
Four-byte input parameter that specifies the address of a parameter list used by the request to pass parameters to CQS. The length of the storage area must be at least equal to the EQU value CQSDEREG_PARM_LEN (defined using the FUNC=DSECT request).

RETCODE=returncodeaddress
Output parameter that specifies the address of a 4-byte field to contain the CQSDEREG return code. The CQSDEREG return code is returned both in this field and in register 15.

RSNCODE=reasoncodeaddress
Output parameter that specifies the address of a 4-byte field to contain the CQSDEREG reason code. The CQSDEREG reason code is returned both in this field and in register 0.

Return and Reason Codes for CQSDEREG
Table 33 lists the return and reason code combinations that can be returned for CQSDEREG requests.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000000'</td>
<td>X'00000000'</td>
<td>Request completed successfully.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000010'</td>
<td>Unable to free CQS's storage in client's address space. The cqstoken is now invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000012'</td>
<td>The CQSDEREG parameter list version is invalid. This error is probably caused by a difference in versions between the CQS client and the CQS address space the client is trying to use.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000028'</td>
<td>Parmlist version is invalid.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000043'</td>
<td>Request is active.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000050'</td>
<td>CQS internal error. The cqstoken is now invalid.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000050'</td>
<td>Storage allocation error for work area.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000051'</td>
<td>CQS internal error (unable to create ESTAE).</td>
</tr>
</tbody>
</table>

CQSDISC Request

Format for CQSDISC

DISCABND Function of CQSDISC: Use the DISCABND function of a CQSDISC request while the client is terminating abnormally to terminate client connections to all coupling facility structures.
**DISCNORM Function of CQSDISC:** Use the DISCNORM function of a CQSDISC request while the client is terminating normally to terminate client connections to one or more coupling facility structures.

**DSECT Function of CQSDISC:** Use the DSECT function of a CQSDISC request to include equate (EQU) statements in your program for the CQSDISC parameter list length, CQSDISC return and reason codes, and literals that can be used to build the OPTWORD1 parameter.

**Usage of CQSDISC**

**Restriction:** The CQSDISC request does not support structure attributes for resource structures.
The CQSDISC request allows a client to disconnect from one or more coupling facility structures. CQS disconnects client resources associated with the structures. The client needs to issue a CQSDEREG request to completely disconnect from CQS.

A CQSDISC FUNC=DISCABND request, used when the client is terminating abnormally, terminates client connections to all coupling facility structures.

A CQSDISC FUNC=DISCNORM, used when the client is terminating normally, terminates client connections to one or more coupling facility structures. The client specifies a disconnect list containing one or more list entries, for which each entry is a separate disconnect request. As each structure disconnect is completed, the connect token for that structure is invalidated and can no longer be used by the client.

Parameter Description:

**COUNT=count**

Four-byte input parameter that specifies the number of list entries in the disconnect list.

**CQSSHUT=YES | NO**

Input parameter that indicates whether or not the CQS address space should be shut down after all clients have disconnected.

If CQSSHUT=YES is specified, new clients continue to be allowed to issue CQSCONN requests. The CQSSHUT FUNC=QUIESCE request can be used to prevent new clients from issuing CQSCONN requests.

The CQSSHUT parameter cannot be used when the OPTWORD1 parameter is specified. If you specify OPTWORD1 instead of CQSSHUT, you can use the following equate (EQU) symbols to generate the value for the OPTWORD1 parameter:

- CQSDISC_SHUTYEQX
- CQSDISC_SHUTNEQX

**CQSTOKEN=cqstokenaddress**

Input parameter that specifies the address of the 16-byte CQS registration token that uniquely identifies the client’s connection to CQS. The registration token is returned by the CQSREG request.

**ECB=ecbaddress**

Four-byte input parameter that specifies the address of the MVS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise it is processed synchronously.

**LIST=listaddress**

Four-byte input parameter that specifies the address of a disconnect list containing one or more entries. Each entry is a separate request to disconnect a client from a coupling facility structure. Some fields in each entry must be initialized by the client prior to the CQSDISC request. Other fields are returned by CQS upon completion of the CQSDISC request.

The CQSDISCL list entry DSECT maps the list entries and can be used by the client. Multiple list entries must reside in contiguous storage.

Each list entry contains the following:

**connecttoken**

Sixteen-byte input parameter that specifies the connect token that
uniquely identifies the client’s connection to a particular coupling facility structure managed by this CQS. The connect token is returned by the CQSCONN request. This parameter is required.

structureattributes
Four-byte input parameter field that contains the structure attributes.

+0  Flag byte 1, with the following bits defined:

X'80'
When set to 0, indicates that CQS should not perform a structure checkpoint for the structure.

When set to 1, indicates that CQS should perform a structure checkpoint for the structure.

X'40'
When set to 0, indicates that CQS should not perform disconnect processing for the structure if there is any inflight work (locked objects) on the structure. If inflight work is found, CQS will set completion code X'00000008' in the compcode field, and will return a return code of X'0000000C', and a reason code of either X'00000300' or X'00000304' for the request.

When set to 1, indicates that CQS should disconnect from the structure, even if there is inflight work (locked objects) on the structure. If inflight work is found, CQS will set completion code X'00000008' in the compcode field, and will return a return code of X'00000004', and a reason code of X'00000140' for the request, if no other errors in disconnect processing occur. Note that the return and reason code is a warning only; the disconnect processing is still performed.

The remaining bits in this byte are not used, and must be set to zero.

+1  The next three bytes are not used, and must be set to zero.

compcode
Four-byte output field to receive the completion code from the request. Possible completion codes are:

X'00000000'
Request completed successfully.

X'00000004'
connecttoken is invalid.

X'00000008'
The client has inflight work for the structure. If the X'40' bit in the first byte of the structureattributes parameter was set to one, the disconnect processing was successful for the structure, and this completion code is informational.
If the X'40' bit was zero, the disconnect processing was not done for this structure, and the CQS client should complete the inflight work before continuing.

X'0000000C'
Structure attributes are not allowed for a resource structure.

LISTVER=1 | listversion
Input parameter that specifies an equate for the list version. Use the DSECT function of a CQSDISC request to include equate (EQU) statements in your program for the CQSDISC list versions.

OPTWORD1=optionwordvalue
Four-byte input parameter that specifies the literals for this request. This parameter can be used instead of CQSSHUT. Equate (EQU) statements for the literal values are listed under the description of the CQSSHUT parameter. Equate statements can also be generated by using the DSECT function. The OPTWORD1 parameter cannot be used if CQSSHUT is specified.

Requirement: If you code the OPTWORD1 parameter, you must pass a value that is composed of one equate value for each literal value supported by this macro.

PARM=parmaddress
Four-byte input parameter that specifies the address of a parameter list used by the request to pass parameters to CQS. The length of the storage area must be at least equal to the EQU value CQSDISC_PARM_LEN (defined using the FUNC=DSECT request).

RETCODE=returncodeaddress
Output parameter that specifies the address of a 4-byte field to contain the CQSDISC return code.

If the return code in register 15 is nonzero, the values in the return and reason code fields are invalid, because the CQS interface detected an error and was unable to send the request to CQS.

RSNCODE=reasoncodeaddress
Output parameter that specifies the address of a 4-byte field to contain the CQSDISC reason code.

Return and Reason Codes for CQSDISC

Table 34 lists the return and reason code combinations that can be returned for CQSDISC requests. Use a CQSDISC FUNC=DSECT request to include equate statements in your program for the return and reason codes.

Table 34. CQSDISC Return and Reason Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000000'</td>
<td>X'00000000'</td>
<td>Request completed successfully.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000130'</td>
<td>Request completed successfully for the requested structures. Client is still connected to additional coupling facility structures.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000140'</td>
<td>Request completed successfully for the requested structures. At least one structure had inflight work for this client, but the client indicated that disconnect processing was allowed with inflight work at CQSDISC. The completion code field for those structures contains X'00000008'.</td>
</tr>
</tbody>
</table>
Table 34. CQSDISC Return and Reason Codes (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000008'</td>
<td>X'00000210'</td>
<td>cqstoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000218'</td>
<td>FUNC is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000250'</td>
<td>count is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000254'</td>
<td>listaddress is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000284'</td>
<td>Parmlist version is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000288'</td>
<td>List version is invalid.</td>
</tr>
<tr>
<td>X'0000000C'</td>
<td>X'00000300'</td>
<td>Request succeeded for at least one but not all list entries. See compcode for individual errors.</td>
</tr>
<tr>
<td>X'0000000C'</td>
<td>X'00000304'</td>
<td>Request failed for all list entries. See compcode for individual errors.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000430'</td>
<td>No CQS address space.</td>
</tr>
</tbody>
</table>

CQSINFRM Request

Format for CQSINFRM

*DSECT Function of CQSINFRM:* Use the DSECT function of a CQSINFRM request to include equate (EQU) statements in your program for the CQSINFRM parameter list length and CQSINFRM return and reason codes.

```
CQSINFRM—FUNC=DSECT
```

*INFORM Function of CQSINFRM:* Use the INFORM function of a CQSINFRM request to register a client’s interest in one or more queues on a specific coupling facility structure.

```
CQSINFRM—FUNC=INFORM—CQSTOKEN=cqstokenaddress—PARAM=parmaddress
```

```
COUNT=count—LISTSIZE=listsize—LIST=listaddress
```

```
ECB=ecbaddress
```

```
RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress
```

```
LISTVER=LISTVER=listversion
```

*UNINFORM Function of CQSINFRM:* Use the UNINFORM function of a CQSINFRM request to deregister a client’s interest in one or more queues on a specific coupling facility structure it previously registered interest for.
Usage of CQSINFRM
A client uses a CQSINFRM request to register or deregister interest for one or more queues on a specific coupling facility structure. When a queue goes from empty to non-empty, CQS notifies all clients that registered interest for the queue of the change in status by scheduling the Structure Inform Client exit routine.

Restriction: The CQSINFRM request is not supported for resource structures.

Related Reading: For more information on the Structure Inform Client exit routine, see "CQS Client Structure Inform Exit Routine" on page 181.

The client can issue CQSREAD or CQSBRWSE requests to retrieve data from a queue. A client can make data objects available on a queue using CQSPUT, CQSMOVE, or CQSUNLCK requests.

A client that has registered interest in a queue is only notified when the queue goes from empty to non-empty, or if a data object is available on the queue when the CQSINFRM request is issued. The client does not receive notification when additional data objects are placed on a non-empty queue.

After a client deregisters interest in a queue, it is no longer notified when one of the queues goes from empty to non-empty. Because client notifications occur asynchronously with CQSINFRM requests, the client should expect to be notified about new data objects that arrive between the time the client issues the CQSINFRM FUNC=UNINFORM request and the time CQS processes the request.

Parameter Description:

COUNT=count
Four-byte input parameter that specifies the number of structure list entries in the structure list.

CQSTOKEN=cqstokenaddress
Input parameter that specifies the address of the 16-byte CQS registration token that uniquely identifies the client’s connection to CQS. The registration token is returned by the CQSREG request.

ECB=ecbaddress
Four-byte input parameter that specifies the address of the MVS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise it is processed synchronously.
LIST=listaddress

Four-byte input parameter that specifies the address of the structure list. The structure list is built in contiguous storage, and the size of the list must be specified using the LISTSIZE parameter. The structure list should contain an entry for each coupling facility structure for which the client will register or deregister interest. Each structure list entry must contain a list of the queues for which the client will register or deregister interest.

Each connect token in a structure list entry and queue name in the queue list entry must be initialized prior to the request. Upon completion of the request, CQS returns the structure completion code for the structure list and the queue completion code for the queue list.

The CQSINFL list entry DSECT maps the queue and structure list entries and can be used by the client. Multiple list entries must reside in contiguous storage.

Each structure list entry contains the following:

connecttoken

Sixteen-byte input parameter that specifies the connect token that uniquely identifies the client’s connection to CQS and a specific coupling facility structure. The connect token is returned by the CQSCONN request. This parameter is required.

structurecompletioncode

Four-byte output field to receive the completion code for the CQSINFRM request for the structure. Possible structure completion codes are:

- X'00000000' Request completed successfully.
- X'00000004' Request completed successfully for all queues. At least one queue has work on it. See the queue completion code to determine which queues have work on them.
- X'00000010' connecttoken is invalid.
- X'00000014' queuelistcount is invalid.
- X'00000018' Inform exit routine does not exist. The Structure Inform exit routine was not specified on CQSCONN request for structure.
- X'00000020' Request completed successfully for at least one, but not all queues in queuelist. See queuecompletioncode for individual errors.
- X'00000024' Request failed for all queues in queuelist. See queuecompletioncode for individual errors or successes.
- X'00000030' A CQSSYNC is required for this structure.
- X'00000034' CQSINFRM is not allowed for a resource structure.

queuelistcount

Four-byte input parameter that specifies the number of queues in the queue list. This parameter is required.

Recommendation: For optimum performance, a client that registers interest in many queues should issue multiple CQSINFRM requests, in which each request lists no more than 1024 queues.
queuelist
Variable length input area that contains one or more queue lists. A
queue list, built by the client, should contain an entry for each
queue on the structure for which the client will register or deregister
interest. The queue names must be initialized prior to the request.
This parameter is required.

Each queue list entry contains the following:

queuename
Sixteen-byte input field that contains the name of the queue
for which the client is registering interest. This parameter is
required.

queuerequestflag
One-byte input field that contains flags specific to this
queue that can be set for this CQSINFRM request.

X'80'
Call the client Inform exit routine if there are
data objects on the queue at the time the
client issues the CQSINFRM
FUNC=INFORM request. Applies only to
CQSINFRM FUNC=INFORM requests.

queuecompletioncode
Four-byte output field to receive the completion code for the
specified queue. Possible completion codes are:

X'00000000'
Request completed successfully.

X'00000040'
Work exists on queue.

X'00000044'
queuename is invalid.

X'00000048'
CQS internal error.

X'00000050'
Structure is full. No more event monitoring
controls (EMC) are available for queue
registration.

X'00000054'
Structure is inaccessible. Retry request.

LISTSIZE=listsize
Four-byte input parameter that specifies the size of the structure list. The
client builds the structure list and must specify the size of the structure list
in this field.

LISTVER=1 | listversion
Input parameter that specifies an equate for the list version. Use the
DSECT function of a CQSINFRM request to include equate (EQU)
statements in your program for the CQSINFRM list versions.

PARM=parmaddress
Four-byte input parameter that specifies the address of a parameter list
used by the request to pass parameters to CQS. The length of the storage
area must be at least equal to the EQU value CQSINFRM_PARM_LEN
(defined using the FUNC=DSECT request).

RETCODE=returncodeaddress
Output parameter that specifies the address of a 4-byte field to contain the
CQSINFRM return code.
If the return code in register 15 is nonzero, the values in the return and reason code fields are invalid, because the CQS interface detected an error and was unable to send the request to CQS.

\[ \text{RSNCODE} = \text{reasoncodeaddress} \]
Output parameter that specifies the address of a 4-byte field to contain the CQSINFRM reason code.

**Return and Reason Codes for CQSINFRM**

Table 35 lists the return and reason code combinations that can be returned for CQSINFRM requests. Use a CQSINFRM FUNC=DSECT request to include equate statements in your program for the return and reason codes.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000000'</td>
<td>X'00000000'</td>
<td>Request completed successfully.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000134'</td>
<td>Request completed successfully. One or more queues have work.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000210'</td>
<td>cqstoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000218'</td>
<td>FUNC is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000250'</td>
<td>count is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000254'</td>
<td>listaddress is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000258'</td>
<td>listsize is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000284'</td>
<td>Parmlist version is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000288'</td>
<td>List version is invalid.</td>
</tr>
<tr>
<td>X'0000000C'</td>
<td>X'00000300'</td>
<td>Request succeeded for at least one, but not all, list entries. Check \text{structurecompletioncode} for individual errors or successes.</td>
</tr>
<tr>
<td>X'0000000C'</td>
<td>X'00000304'</td>
<td>Request failed for all list entries. See \text{structurecompletioncode} for individual errors.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000430'</td>
<td>No CQS address space.</td>
</tr>
</tbody>
</table>

**CQSMOVE Request**

**Format for CQSMOVE**

\textit{DSECT Function of CQSMOVE:} Use the DSECT function of a CQSMOVE request to include equate (\texttt{EQU}) statements in your program for the CQSMOVE parameter list length, CQSMOVE return and reason codes, and literals that can be used to build the \texttt{OPTWORD1} parameter.

\[ \text{CQSMOVE—FUNC=DSECT} \]

\textit{MOVE Function of CQSMOVE:} Use the MOVE function of a CQSMOVE request to move one or all data objects from one queue to another. You must code a macro invocation for each combination of literal parameters.
You can use the OPTWORD1 parameter to code a single invocation of the macro and set the options at runtime. However, you cannot use the COUNT, NEWQPOS, and OLDQPOS parameters if you use the OPTWORD1 parameter.

**MOVE Function of CQSMOVE using Literal Parameters**

![Diagram of CQSMOVE using Literal Parameters]

**A:**

- COUNT=ONE
- COUNT=ALL
- MVCNT=movecountaddress

You can use the OPTWORD1 parameter to code a single invocation of the macro and set the options at runtime. However, you cannot use the COUNT, NEWQPOS, and OLDQPOS parameters if you use the OPTWORD1 parameter.

**MOVE Function of CQSMOVE using OPTWORD1 Parameter**

![Diagram of CQSMOVE using OPTWORD1 Parameter]

**Usage of CQSMOVE**

**Restriction:** The CQSMOVE request is not supported for resource structures.

A CQSMOVE request moves one or all client data objects from one queue to another. Data objects can be moved from the first or last position of the old queue to the first or last position on the new queue. The client identifies the data objects to be moved either by the old queue name and queue position, or by the lock token. Do not move multiple objects with the same queue name and UOW; otherwise CQS cannot recover the objects.

If CQS or the client fails before CQS responds to the client, the CQSMOVE request might not complete. The client must reconnect to CQS after the failure and may have to issue the CQSMOVE request again, in case the failure occurred before the move was committed, or to resume a move with COUNT=ALL.
Parameter Description:

**CONTOKEN=connecttokenaddress**
Input parameter that specifies the address of the 16-byte connect token that uniquely identifies the client's connection to a particular coupling facility structure managed by this CQS. The connect token is returned by the CQSCONN request.

**COUNT=ONE | ALL**
Input parameter that specifies the number of data objects on the old queue to be moved; the client can move either one or all of them.

The COUNT parameter cannot be used when the OPTWORD1 parameter is specified. If you specify the OPTWORD1 parameter instead of the COUNT parameter, you can use the following equate (EQU) symbols to generate the value for the OPTWORD1 parameter:

```
CQSMOVE_CNT1EQUX COUNT=ONE
CQSMOVE_CNT1EQUX COUNT=ALL
```

**CQSTOKEN=cqstokenaddress**
Input parameter that specifies the address of the 16-byte CQS registration token that uniquely identifies the client's connection to CQS. The registration token is returned by the CQSREG request.

**ECB=ecbaddress**
Four-byte input parameter that specifies the address of the MVS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise it is processed synchronously.

**LCKTOKEN=locktokenaddress**
Input parameter that specifies the address of the 16-byte lock token for the locked data object to be moved. The lock token uniquely identifies a data object locked by a CQSREAD request.

**MVCNT=movecountaddress**
Output parameter that specifies the address of a 4-byte field to receive the number of data objects that were moved. Even when the return or reason code is non-zero, it is possible that CQS moved some data objects.

**NEWQ=newqueuenameaddress**
Input parameter that specifies the address of the 16-byte name of the new queue to which the data object is to be moved.

**NEWQPOS=FIRST | LAST**
Input parameter that specifies the position on the new queue to which data objects are moved, either first or last.

The NEWQPOS parameter cannot be used when the OPTWORD1 parameter is specified. If the OPTWORD1 parameter is specified instead of NEWQPOS, you can use the following equate (EQU) statements to generate the value for the OPTWORD1 parameter:

```
CQSMOVE_NEWQFEQUX NEWQPOS=FIRST
CQSMOVE_NEWQLEQUX NEWQPOS=LAST
```

**OLDQ=oldqueuenameaddress**
Input parameter that specifies the address of the 16-byte name of the old queue from which the data object is to be moved.

**OLDQPOS=FIRST | LAST**
Input parameter that specifies the position on the old queue from which data objects are to be moved, either first or last.
The OLDQPOS parameter cannot be used when the OPTWORD1 parameter is specified. If the OPTWORD1 parameter is specified instead of OLDQPOS, you can use the following equate (EQU) statements to generate the value for the OPTWORD1 parameter:

```
CQSMOVE_OLDQFEQUX  OLDQPOS=FIRST
CQSMOVE_OLDQLEQUX  OLDQPOS=LAST
```

**OPTWORD1=optionvalue**

Four-byte input parameter that specifies the literals for this request. This parameter can be used instead of COUNT, NEWQPOS, and OLDQPOS. Equate (EQU) statements for the literal values are listed under the COUNT, NEWQPOS, and OLDQPOS parameter descriptions. Equate statements can also be generated by using the DSECT function. The OPTWORD1 parameter cannot be used if COUNT, NEWQPOS, or OLDQPOS is specified.

**Requirement:** If you code the OPTWORD1 parameter, you must pass a value that is composed of one equate value for each literal value supported by this macro.

**PARM=parmaddress**

Four-byte input parameter that specifies the address of a parameter list used by the request to pass parameters to CQS. The length of the storage area must be at least equal to the EQU value CQSMOVE_PARM_LEN (defined using the FUNC=DSECT request).

**RETCODE=returncodeaddress**

Output parameter that specifies the address of a 4-byte field to contain the CQSMOVE return code.

If the return code in register 15 is nonzero, the values in the return and reason code fields are invalid, because the CQS interface detected an error and was unable to send the request to CQS.

**RSNCODE=reasoncodeaddress**

Output parameter that specifies the address of a 4-byte field to contain the CQSMOVE reason code.

### Return and Reason Codes for CQSMOVE

Table 36 lists the return and reason code combinations that can be returned for CQSMOVE requests. Use a CQSMOVE FUNC=DSECT request to include equate statements in your program for the return and reason codes.

**Table 36. CQSMOVE Return and Reason Codes**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000000'</td>
<td>X'00000000'</td>
<td>Request completed successfully.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'000000128'</td>
<td>No data object to move for queue name specified.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'000000210'</td>
<td>cqstoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'000000214'</td>
<td>connecttoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'000000218'</td>
<td>FUNC is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000021C'</td>
<td>locktoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'000000220'</td>
<td>Queue name is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'000000224'</td>
<td>Buffer address is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000027C'</td>
<td>CQSMOVE is not allowed for a resource structure.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'000000284'</td>
<td>Parmlist version is invalid.</td>
</tr>
</tbody>
</table>
Table 36. CQSMOVE Return and Reason Codes (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X’00000010’</td>
<td>X’00000400’</td>
<td>A CQSRSYNC is required for this structure.</td>
</tr>
<tr>
<td>X’00000010’</td>
<td>X’00000404’</td>
<td>Structure is inaccessible. Retry request later.</td>
</tr>
<tr>
<td>X’00000010’</td>
<td>X’00000414’</td>
<td>Unable to move the data object because the destination queue is full. CQSMOVE requests for other queues are allowed.</td>
</tr>
<tr>
<td>X’00000010’</td>
<td>X’0000041C’</td>
<td>Request pending. A structure recovery or CQS restart might be required to complete.</td>
</tr>
<tr>
<td>X’00000010’</td>
<td>X’00000430’</td>
<td>No CQS address space.</td>
</tr>
<tr>
<td>X’00000010’</td>
<td>X’00000440’</td>
<td>Locked (nonrecoverable) data object lost due to rebuild.</td>
</tr>
<tr>
<td>X’00000014’</td>
<td>X’00000500’</td>
<td>CQS internal error.</td>
</tr>
</tbody>
</table>

CQSPUT Request

Format for CQSPUT

**ABORT Function of CQSPUT:** Use the ABORT function of a CQSPUT request to remove from the queues all uncommitted data objects associated with a recoverable unit of work.

```
►►CQSPUT—FUNC=ABORT—CQSTOKEN=cqstokenaddress—CONTOKEN=connecttokenaddress—PN=parmaddress—PUTTOKEN=puttokenaddress—ECB=ecbaddress—RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress
```

**DSECT Function of CQSPUT:** Use the DSECT function of a CQSPUT request to include equate (EQU) statements in your program for the CQSPUT parameter list length, CQSPUT return and reason codes, and literals that can be used to build the OPTWORD1 parameter.

```
►►CQSPUT—FUNC=DSECT—PN=parmaddress—PUTTOKEN=puttokenaddress—ECB=ecbaddress—RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress
```

**FORGET Function of CQSPUT:** Use the FORGET function of a CQSPUT request to discard any information CQS has on a committed unit of work.

```
►►CQSPUT—FUNC=FORGET—CQSTOKEN=cqstokenaddress—CONTOKEN=connecttokenaddress—PN=parmaddress—PUTTOKEN=puttokenaddress—ECB=ecbaddress—RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress
```

**PUT Function of CQSPUT:** Use the PUT function of a CQSPUT request to place a data object on a queue.

```
►►CQSPUT—FUNC=PUT—CQSTOKEN=cqstokenaddress—CONTOKEN=connecttokenaddress—PN=parmaddress—PUTTOKEN=puttokenaddress—ECB=ecbaddress—RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress
```
Usage of CQSPUT

**Restriction:** The CQSPUT request is not supported for resource structures.

A CQSPUT request allows a client to place a data object on a queue. The data object can be either the only one for a unit of work, or it can be one in a series for a unit of work. The data object can be added to the beginning or to the end of the queue. After the data object is on the queue, it is available to any client that has access to that queue.
You can put multiple objects on the same queue for unit of work. Do not move these objects (CQSMOVE request) or allow these objects to be moved to the overflow structure (CQSCONN request); otherwise, CQS cannot recover the objects.

If a unit of work consists of multiple data objects, and they are all on the same queue, then when CQS places the first data object on the queue, it notifies other clients that have registered interest in the queue, even though not all of the data objects for the UOW are on the queue yet and the UOW has not yet been committed.

**Recommendation:** To ensure that a client does not retrieve incomplete data, place the last data object for a UOW on a different queue than any of the previous data objects for the unit of work, and ensure the client only registers interest in that queue.

The first request that places a data object on a queue for a unit of work determines whether that unit of work is recoverable or nonrecoverable. The actions taken for a data object when a client fails, CQS fails, a structure is copied, or a structure is recovered depend on whether the unit of work is recoverable and, if so, whether it has been committed. Table 37 shows the actions taken for each case.

When a message is put on the shared queues, a timestamp indicating when the message is put on the queue is saved in the CQS prefix of the message. If TIMESTAMP= is then specified on the CQSPUT request, the CQSPUT timestamp value is used. If TIMESTAMP= is not specified, CQS generates a timestamp indicating when the message is put on the shared queue. This generated timestamp is returned on the CQSQUERY FUNC=QTYPE request.

| Table 37. Actions Taken for Data Objects as a Result of Failures or Structure Activity |
|-----------------------------------|-------------------------------|-------------------------------|
| **Nonrecoverable**                | **Recoverable and Uncommitted** | **Recoverable and Committed** |
| **Client Failure**                | All data objects on the queues for nonrecoverable units of work are left on the queues. | All data objects on the queues that belong to uncommitted units of work are deleted when the client terminates. | All data objects on the queues for the unit of work remain on the queues. |
| **CQS Failure**                   | Any data objects for non-recoverable units of work that were placed on the queues successfully are left on the queues. If CQS was in the process of placing a data object on a queue when the failure occurred, that data object is not recovered when CQS restarts. | All data objects on the queues that belong to uncommitted units of work are deleted when CQS restarts. | All data objects on the queues that belong to committed units of work remain on the queues. If CQS was in the process of placing the final data object for the unit of work on the queues when the failure occurred, CQS restart ensures the data object is on the queues. |
| **Structure Copy**                | Data objects for non-recoverable units of work are copied to the new structure. | All data objects for recoverable units of work are copied to the new structure whether the unit of work is committed or not. | All data objects for recoverable units of work are copied to the new structure. |
| **Structure Recovery**            | Data objects placed on the queues for nonrecoverable units of work are not recovered to the new structure. | All data objects that were placed on the queues for recoverable units of work are recovered to the new structure whether or not the unit of work was committed. | All data objects that were placed on the queues for recoverable units of work are recovered to the new structure. |
A CQSPUT FUNC=FORGET request terminates any CQSPUT FUNC=PUT requests, and causes CQS to discard internal information CQS has about the unit of work. The unit of work is identified by the put token. The client should make this request after receiving a response from the final CQSPUT FUNC=PUT request issued for the unit of work. The CQSPUT FUNC=FORGET request is rejected if the unit of work is recoverable but not committed.

A CQSPUT FUNC=ABORT request removes from the queues all uncommitted data objects associated with a recoverable unit of work. The unit of work is identified by the put token. The request is rejected if the unit of work is nonrecoverable or if the unit of work is recoverable, but already committed.

Examples: To put a single object for a unit of work on the queues, issue the following requests:

```
CQSPUT FUNC=PUT,COMMIT=YES,...
...
CQSPUT FUNC=FORGET,...
```

To put multiple objects for a unit of work on the queues, issue the following requests:

```
CQSPUT FUNC=PUT,COMMIT=NO,...
...
CQSPUT FUNC=PUT,COMMIT=NO,...
...
CQSPUT FUNC=PUT,COMMIT=YES,...
...
CQSPUT FUNC=FORGET,...
```

Parameter Description:

**COMMIT=YES | NO**

Input parameter that indicates whether to commit a recoverable unit of work. One or more data objects can be placed on the queues for a recoverable unit of work.

The COMMIT= parameter applies only to recoverable units of work and is only valid if RECOVERABLE=YES is specified. The parameter is ignored if RECOVERABLE=NO is specified.

COMMIT=YES must be specified (either by itself or as part of OPTWORD1) for the final (or only) CQSPUT FUNC=PUT request issued for a unit of work. If more than one data object is placed on the queues for a unit of work, COMMIT=NO must be specified on all except the final CQSPUT FUNC=PUT request of the series. COMMIT=YES must be specified on the final CQSPUT FUNC=PUT request.

The COMMIT parameter cannot be used if the OPTWORD1 parameter is specified. If the OPTWORD1 parameter is used instead of COMMIT, you can use the following equate (EQU) statements to generate the value for the OPTWORD1 parameter:

- `CQSPUT_CMTYEQUX COMMIT=YES`
- `CQSPUT_CMTNEQUX COMMIT=NO`

**CONTOKEN=connecttokenaddress**

Input parameter that specifies the address of the 16-byte connect token that uniquely identifies the client's connection to a particular coupling facility structure managed by this CQS. The connect token is returned by the CQSCONN request.
CQSTOKEN=\textit{cqstokenaddress}

Input parameter that specifies the address of the 16-byte CQS registration token that uniquely identifies the client’s connection to CQS. The registration token is returned by the CQSREG request.

\textbf{DATAOBJ=}\textit{dataobjectaddress}

Four-byte input parameter that specifies the address of the client data object to be placed on the specified queue.

\textbf{ECB=}\textit{ecbaddress}

Four-byte input parameter that specifies the address of the MVS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise it is processed synchronously.

\textbf{LOCAL=}\text{NO | N | YES | Y}

Input parameter that indicates whether the client should keep a local copy of the data.

\textbf{NO}

Indicates the client wants CQS to place the data object on the specified client queue and make the object available to other CQSs.

\textbf{YES}

Indicates that the client wants CQS to place the data object on the shared queues and to lock the object. LOCAL=YES also indicates that the client will keep a local copy of the data object in a local buffer.

By keeping a local copy of the data object, the client can reduce the performance overhead of using shared queues. By keeping the data object on the shared queues, it can be recovered if the client fails. By locking the data object, it is not available to any other client.

The client must issue the CQSREAD LOCAL=YES request to process the data (retrieve the lock token for the data object and inform CQS that the client is processing the data). The data object is not returned to the client on a CQSREAD request because the client has the local copy. If the client does not issue the CQSREAD LOCAL=YES request and the connection between the client and CQS is lost, CQS unlocks the data object and makes it available to any client.

The LOCAL parameter cannot be used if the OPTWORD1 parameter is specified. If the OPTWORD1 parameter is used instead of LOCAL, you can use the following equate (EQU) statements to generate the value for the OPTWORD1 parameter:

\begin{align*}
\text{CQSPUT\_LCLYEQUX} & \quad \text{LOCAL=YES} \\
\text{CQSPUT\_LCLNEQUX} & \quad \text{LOCAL=NO}
\end{align*}

\textbf{OBJSIZE=}\textit{dataobjectsizedaddress}

Input parameter that specifies the address of a 4-byte area to hold the size of the client data object to be placed on the queue. The maximum size that can be specified is 61312 bytes (X’EF80’).

\textbf{OPTWORD1=}\textit{optionwordvalue}

Four-byte input parameter that specifies the literals for this request. This parameter can be used instead of COMMIT, LOCAL, QPOS, and RECOVERABLE. Equate (EQU) statements for the literal values are listed under the descriptions of the COMMIT, LOCAL, QPOS, and RECOVERABLE parameters. Equate statements can be also generated by
using the DSECT function. The OPTWORD1 parameter cannot be used if COMMIT, LOCAL, QPOS or RECOVERABLE is specified.

**Requirement:** If you code the OPTWORD1 parameter, you must pass a value that is composed of one equate value for each literal value supported by this macro.

**PARM=parmaddress**

Four-byte input parameter that specifies the address of a parameter list used by the request to pass parameters to CQS. The length of the storage area must be at least equal to the `EQU` value CQSPUT_PARM_LEN (defined using the FUNC=DSECT request).

**PUTTOKEN=puttokenaddress**

Four-byte input and output parameter that specifies the address of a 16-byte token to be used by CQS to relate a series of CQSPUT requests for a unit of work. The token must be zero for the initial CQSPUT request of a series. An updated token is returned by CQS for each CQSPUT request. The updated token must be returned to CQS on the next CQSPUT request for the unit of work. The `puttoken` must also be returned to CQS for any CQSPUT FUNC=FORGET or CQSPUT FUNC=ABORT requests.

**QNAME=queuenameaddress**

Input parameter that specifies the address of the 16-byte name of the queue on which the data object is to be placed. The first byte of the queue name cannot be zero; it is used to determine the queue type. If the value in the first byte is greater than the maximum number of queue types defined by CQS, it is folded into one of the existing queue types. If the last data object for a unit of work is being put on the structure, the data object must be put on a different queue than any of the previous data objects for that unit of work.

**QPOS=LAST | FIRST**

Input parameter that specifies the position on the queue at which to place the client data object.

- **LAST** The data object is added to the end of the queue.
- **FIRST** The data object is added to the beginning of the queue.

The QPOS parameter cannot be used if the OPTWORD1 parameter is specified. If the OPTWORD1 parameter is specified instead of QPOS, you can use the following equate (`EQU`) statements to generate the value for the OPTWORD1 parameter:

- `CQSPUT_QPOSFEQUX` QPOS=FIRST
- `CQSPUT_QPOSLEQUX` QPOS=LAST

**RECOVERABLE=YES | NO**

Input parameter that specifies whether the unit of work is recoverable by CQS. RECOVERABLE=NO indicates the unit of work is nonrecoverable. Only one data object can be placed on the queues for a nonrecoverable unit of work. RECOVERABLE=YES indicates the unit of work is recoverable. One or more data objects can be placed on the queues for a recoverable unit of work.

The RECOVERABLE=YES parameter must be specified for each CQSPUT FUNC=PUT request issued for the unit of work. The unit of work is not committed until the final (or only) data object for the series is placed on the queues (COMMIT=YES specified).
The RECOVERABLE parameter cannot be used if the OPTWORD1 parameter is specified. If the OPTWORD1 parameter is specified instead of RECOVERABLE, you can use the following equate (EQU) statements to generate the value for the OPTWORD1 parameter:

```
CQSPUT_RECVYEQUX RECOVERABLE=YES
CQSPUT_RECVNEQUX RECOVERABLE=NO
```

**RETCODE=returncodeaddress**

Output parameter that specifies the address of a 4-byte field to contain the CQSPUT return code.

If the return code in register 15 is nonzero, the values in the return and reason code fields are invalid, because the CQS interface detected an error and was unable to send the request to CQS.

**RSNCODE=reasoncodeaddress**

Output parameter that specifies the address of a 4-byte field to contain the CQSPUT reason code.

**TIMESTAMP=timestampaddress**

Four-byte input parameter that specifies the address of an 8-byte STCK value that is stored with the data object as the time the data object was placed on the queue. If the TIMESTAMP parameter is omitted, the current time is stored with the data object.

**UOW=uowaddress**

Input parameter that specifies the address of a 32-byte area to hold the unit of work. This parameter is required for the initial (or only) CQSPUT FUNC=PUT request issued for a unit of work. It is ignored for all subsequent CQSPUT FUNC=PUT requests issued for that unit of work.

When a value is specified for the UOW= parameter, PUTTOKEN=0 must also be specified. The value specified for the UOW= parameter cannot be all zeroes, and must be unique within the shared queues. The client is responsible for ensuring that the value is unique.

### Return and Reason Codes for CQSPUT

Table 38 lists the return and reason code combinations that can be returned for CQSPUT requests. Use a CQSPUT FUNC=DSECT request to include equate statements in your program for the return and reason codes.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000000'</td>
<td>X'00000000'</td>
<td>Request completed successfully.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000210'</td>
<td>cqstoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000214'</td>
<td>connecttoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000218'</td>
<td>FUNC is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000220'</td>
<td>puttoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000224'</td>
<td>queue name is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000228'</td>
<td>dataobject is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000230'</td>
<td>uow is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000238'</td>
<td>The queue name is not unique. If more than one data object is placed on the queues for a unit of work, the queue name assigned to the last data object must be unique for that unit of work.</td>
</tr>
</tbody>
</table>
Table 38. CQSPUT Return and Reason Codes (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000008'</td>
<td>X'00000260'</td>
<td>A CQSPUT FUNC=PUT request was issued, but the unit of work was already committed.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000264'</td>
<td>A CQSPUT FUNC=FORGET request was issued for a recoverable unit of work, but the unit of work was not committed.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000268'</td>
<td>A CQSPUT FUNC=ABORT request was issued for a nonrecoverable unit of work.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'0000026C'</td>
<td>A CQSPUT FUNC=ABORT request was issued for a recoverable unit of work but the unit of work was already committed.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000270'</td>
<td>A subsequent CQSPUT FUNC=PUT request was issued for a unit of work already known to CQS as non-recoverable. Only one data object can be placed on the queues for a nonrecoverable unit of work.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000274'</td>
<td>RECOVERABLE=NO was specified for a unit of work that was indicated as recoverable on a previous CQSPUT FUNC=PUT request.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'0000027C'</td>
<td>CQSPUT is not allowed for a resource structure.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000284'</td>
<td>Parmlist version is invalid.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000400'</td>
<td>A CQSRSYNC is required for this structure.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000404'</td>
<td>Structure inaccessible. Retry request later.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000414'</td>
<td>Queue for queue name is full. No more data objects can be inserted to the structure for this queue name. CQSPUT requests for other queue names are still allowed.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000418'</td>
<td>Structure is full. All CQSPUT requests are rejected.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000430'</td>
<td>No CQS address space.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000500'</td>
<td>CQS internal error.</td>
</tr>
</tbody>
</table>

CQSQUERY Request

Format for CQSQUERY

**DSECT Function of CQSQUERY:** Use the DSECT function of a CQSQUERY request to include equate (EQU) statements in your program for the CQSQUERY parameter list length and CQSQUERY return and reason codes.

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CQSQUERY</td>
<td>FUNC=DSECT</td>
<td></td>
</tr>
</tbody>
</table>
```

**QNAME Function of CQSQUERY:** Use the QNAME function of a CQSQUERY request to retrieve information about a specific queue managed by CQS.
CQSQUERY

**QRYOBS Function of CQSQUERY:** Use the QRYOBS function of a CQSQUERY request to retrieve the queue counts for a specified list of queue names.

```c
CQSQUERY FUNC=QNAME CQSTOKEN=token
CONTOKEN=connecttokenaddress PARM=parmaddress COUNT=count
LIST=listaddress ECB=ecbaddress RETCODE=returncodeaddress
RSNCODE=reasoncodeaddress LISTVER=listversion
```

**QTYPE Function of CQSQUERY:** Use the QTYPE function of a CQSQUERY request to retrieve information about all or some of the queues within the specified queue type.

```c
CQSQUERY FUNC=QTYPE CQSTOKEN=token
CONTOKEN=connecttokenaddress PARM=parmaddress COUNT=count
QTYPE=typeaddress QTYPENM=COLDQ
BUFFER=bufferaddress BUFSIZE=buffersize QAGE=queueagevalue
QDATASZ=querydatasizeaddress ECB=ecbaddress RETCODE=returncodeaddress
RSNCODE=reasoncodeaddress LISTVER=listversion
```
**STATISTICS Function of CQSQUERY:** Use the STATISTICS function of a CQSQUERY request to retrieve status information on all the queues managed by CQS.

```
CQSQUERY—FUNC=STATISTICS—CQSTOKEN=cqstokenaddress—PARM=parmaddress—COUNT=count—LIST=listaddress—BUFFER=bufferaddress—BUFSIZE=buffersize—STATSZAR=statisticssizeaddress—ECB=ecbaddress—RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress—LISTVER=listversion
```

**STRSTAT Function of CQSQUERY:** Use the STRSTAT function of the CQSQUERY request to retrieve structure related statistics. The STRSTAT function returns the same statistics data that is given to the Structure Statistics user exit routine.

**Attention:** If the CQS that is processing the request is in the middle of a structure checkpoint, the data returned for the current structure checkpoint might be incomplete.

```
CQSQUERY—FUNC=STRSTAT—CQSTOKEN=cqstokenaddress—CONTOKEN=connecttokenaddress—PARM=parmaddress—BUFFER=bufferaddress—BUFSIZE=buffersize—QDATASZ=querydatasizeaddress—ECB=ecbaddress—RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress—LISTVER=listversion
```

**Usage of CQSQUERY**
The CQSQUERY request retrieves information or status about one or more of the structures managed by CQS. A CQSQUERY FUNC=QNAME request retrieves information about one or more specific queues managed by CQS. A CQSQUERY FUNC=QRYOBJBS request retrieves the queue counts for one or more specific queues or queues whose names match a wildcard parameter. A CQSQUERY FUNC=QTYPE request retrieves information about all or some of the queues within the specified queue type. A CQSQUERY FUNC=STATISTICS request retrieves status information for all queues managed by CQS.
**CQSQUERY**

The `CQSQUERY` request retrieves structure statistics, such as checkpoint and rebuild, without having to code a user exit.

**Restriction:** The `CQSQUERY` `FUNC=QNAME`, `CQSQUERY` `FUNC=QRYOBS`, and `CQSQUERY` `FUNC=QTYPE` requests are not supported for resource structures.

**CQSQUERY** `FUNC=QNAME`: For `CQSQUERY` `FUNC=QNAME`, the number of data objects for the `queueName` specified in `LIST=` is returned.

If the `QAGE` parameter is specified, only information for queues older than the specified queue age is returned. If you are only interested in queue counts, you can omit the `QAGE` parameter for better performance of the `CQSQUERY` request.

**CQSQUERY** `FUNC=QRYOBS`: For `CQSQUERY` `FUNC=QRYOBS`, the number of data objects for the `queueName` specified in `LIST=` is returned. Each queue name in the list can be up to 16 bytes long. The first byte of the `qname` is treated as the `QTYPE`. The input list for each `qname` also has 8 bytes of user data that are copied to the output for each entry that is a match for the input `queueName`.

The `CQSQUERY` `FUNC=QRYOBS` output is returned both in the input list and the output buffer. The input list has the completion code for the `queueName`. If the completion code is 0, then the `queueNames` that match the input `queueName` and their `queueCounts` are returned in the output buffer. If the completion code is non-zero, no data is passed for that queue `name` in the output buffer. The input list has the total `queueCount` found for the `queueName`. If the `queueName` is a wildcard parameter, this `queueCount` is the total `queueCounts` of all the `queueNames` that match the wildcard parameter. An entry for each `queueName` that is a match is passed in the output buffer along with the `queueCount` for the `queueName`. If the buffer size specified is too small, the data that fits in the buffer is passed back, and the actual length required is passed back in the `QDATASZ` field.

**Recommendation:** Use the `CQSQUERY` `FUNC=QRYOBS` request carefully, because it causes CQS to read every data object on the `queueType`, and thus could have a significant performance impact.

**CQSQUERY** `FUNC=QTYPE`: For `CQSQUERY` `FUNC=QTYPE`, information about all the `queueNames` in the `queueType` is returned, including the `queueName`, `dataObjectCount`, oldest data object time stamp, and newest data object time stamp.

**Recommendation:** Use the `CQSQUERY` `FUNC=QTYPE` request carefully, because it causes CQS to read every data object on the `queueType`, and thus could have a significant performance impact.

For `CQSQUERY` `FUNC=QTYPE`, CQS does the following if the buffer area is not large enough to hold all of the requested data:

- Returns as many complete records that can fit into the buffer area
- Sets `QDATASZ` to the length that is needed to contain the statistics data in its entirety
- Sets the reason code for 'Partial Data Returned'

The client program can then make another request with a larger buffer.

**CQSQUERY** `FUNC=STATISTICS`: For `CQSQUERY` `FUNC=STATISTICS`, CQS returns the following information in the client buffer:

- Status on the current capacity of the primary structure
• Maximum capacity of the primary structure (if XES dynamic reconfiguration is available)
• Current operation mode (normal, overflow, or rebuild)
• Elements-to-entries ratio (returned in the buffer passed by the client for this request)

If an overflow structure is defined and the current operation mode for the primary structure is overflow mode, CQS also returns the current and maximum capacity for the associated overflow structure. If the primary structure is not in overflow mode and an overflow structure is defined, CQS returns the overflow structure name and a status indicating that the overflow structure is not in use.

If the buffer area is not large enough to contain the statistics data for all of the requested structures, CQSQUERY FUNC=STATISTICS sets the STATSZAR field to be the length of a single statistics entry, and sets the reason code to 'Buffer Size Too Small.' The size of the buffer that is required to complete the request can be obtained by multiplying the value returned in STATSZAR by the number of list entries specified in the request.

**CQSQUERY FUNC=STRSTAT:** For CQSQUERY FUNC=STRSTAT, CQS returns the following information:
• Structure process statistics
• CQS request statistics
• Data object statistics
• Queue name statistics
• MVS request statistics
• Structure rebuild statistics
• Structure checkpoint statistics

For this function, CQS does the following if the buffer area is not large enough to hold all of the requested data:
• Returns as many complete records that can fit into the buffer area
• Sets QDATASZ to the length that is needed to contain the statistics data in its entirety
• Sets the reason code for 'Partial Data Returned'

The client program can then make another request with a larger buffer.

The following keywords apply to the CQSQUERY macro. Note that some of the information provided below applies to specific CQSQUERY functions.

**BUFFER=bufferaddress**
Four-byte input parameter that specifies the address of the buffer to hold information passed to the client.

For CQSQUERY FUNC=QTYPE, the buffer is mapped by the CQSQRYQT DSECT. For CQSQUERY FUNC=STATISTICS, the buffer is mapped by the CQSQRYST DSECT. For CQSQUERY FUNC=STRSTAT, the buffer is mapped by the CQSQSTAT DSECT. For CQSQUERY FUNC=QRYOBJ, the buffer is mapped by the CQSQRYQO DSECT.

**BUFSIZE=buffersize**
Four-byte input parameter that specifies the size of the buffer passed by the client.
CONTOKEN=connecttokenaddress
Input parameter that specifies the address of the 16-byte connect token that uniquely identifies the client's connection to a particular coupling facility structure managed by this CQS. The connect token is returned by the CQSCONN request.

COUNT= count
Four-byte input parameter that specifies the number of entries in the list.

CQSTOKEN=cqstokenaddress
Input parameter that specifies the address of the 16-byte CQS registration token that uniquely identifies the client's connection to CQS. The registration token is returned by the CQSREG request.

ECB=ecbaddress
Four-byte input parameter that specifies the address of the MVS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise it is processed synchronously.

LIST=listaddress
Four-byte input parameter that specifies the address of a list containing one or more entries. For the CQSQUERY FUNC=QNAME and CQSQUERY FUNC=QRYOBS requests, this list contains queue names for which to retrieve information. The list consists of input and output parameters. At least one list item is required.

The CQSQRYL list entry DSECT maps the list entries and can be used by the client. Multiple list entries must reside in contiguous storage.

For a CQSQUERY FUNC=QNAME request, each list entry contains the following:

compcodedef
Four-byte output field to receive the completion code from the request. Possible completion codes are:
X'00000000' Request completed successfully.
X'00000004' queuename is invalid.
X'00000020' Structure is inaccessible. Retry request.
X'00000024' CQS internal error.

clientdata
Eight-byte input parameter that specifies the client data field. This parameter is optional. CQS does not use data stored in this entry.

queuename
Sixteen-byte input parameter that specifies the queue name for which data object count information is to be retrieved. This parameter is required.

qcnt
Four-byte output parameter that specifies a field to contain the data object count for the queue name specified.

For a CQSQUERY FUNC=STATISTICS request, each list entry contains the following:

compcodedef
Four-byte output field to receive the completion code from the request. Possible completion codes are:
X'00000000'  Request completed successfully.
X'00000008'  connecttoken is invalid.
X'0000000C'  A CQSRSYNC is required for this structure.
X'00000020'  Structure is inaccessible. Retry request.
X'00000024'  CQS internal error.

clientdata
Eight-byte input parameter that specifies the client data field. This
parameter is optional. CQS does not use data stored in this entry.

connecttoken
Sixteen-byte input parameter that specifies the connect token that
uniquely identifies the client’s connection to a particular coupling
facility structure managed by this CQS. The connect token is
returned by the CQSCONN request. This parameter is required.

outputoffset
Four-byte output parameter that specifies the offset of the output
data area for this entry in the output buffer.

For a CQSQUERY FUNC=QRYOBS request, each list entry contains the
following:

compcode
Four-byte output field to receive the completion code from the
request. Possible completion codes are:

X'00000000'  Request completed successfully. A list of resources
that match the qname and their queue counts are
returned in the output buffer.

X'00000004'  qname is invalid.
X'00000010'  qname does not have any objects. The queue
count is zero.
X'00000020'  Retry error for the qname. Retry the CQSQUERY
FUNC=QRYOBS to obtain the queue counts. The
output returned in the output buffer might be invalid.
X'00000024'  CQS internal error. Retry the CQSQUERY
FUNC=QRYOBS to obtain the queue counts. The
output returned in the output buffer might be invalid.

clientdata
Eight-byte input parameter that specifies the client data field. This
parameter is optional. CQS does not use data stored in this entry.

queuename
Sixteen-byte input parameter that specifies the queue name for
which data object count information is to be retrieved. This
parameter is required. The queuename can be a wildcard
parameter.

qcnt
Four-byte output parameter that specifies a field to contain the data
object count for the queue name specified. If the queuename is a
wildcard parameter, this parameter specifies a field to contain the
total queue counts of all qnames that match the wildcard parameter.
LISTVER=1 | listversion
Input parameter that specifies an equate for the list version. Use the DSECT function of a CQSQUERY request to include equate (EQU) statements in your program for the CQSQUERY list versions.

PARM=parmaddress
Four-byte input parameter that specifies the address of a parameter list used by the request to pass parameters to CQS. The length of the storage area must be at least equal to the EQU value CQSQUERY_PARM_LEN (defined using the FUNC=DSECT request).

QAGE=queueageaddress
Input parameter that specifies the address of a 4-byte field to contain the queue age in days. Valid values for queueage are from X'0' to X'16D' (0 to 365 in decimal).

Definition: The queue age is determined by the age of its oldest message, in number of days.

This parameter is used as a filter for determining which queues the CQSQUERY FUNC=QTYPE request will process. The CQSQUERY request returns information for queues containing data objects that are older than the specified queueage. If you specify 0 for queueage, or omit the QAGE parameter, the CQSQUERY request processes all queues for the queue type.

Important: Specifying QAGE causes all the data objects in the queue to be read, which incurs additional performance overhead.

QDATASZ=querydatasizemaddress
Output parameter that specifies the address of a 4-byte field to contain the size of the information returned to the client. If partial data is returned in the buffer, this field contains the actual buffer size needed to hold the information.

QTYPE=queuetypeaddress
Input parameter that specifies the address of a 4-byte field that contains the queue type. Valid values for the queue type are from 1 to 255 (decimal).

QTYPE=M=COLDQ
Input parameter that indicates that the CQSQUERY request is for information about the COLDQ.

This parameter enables a client to obtain the same type of information for the cold queue as can be obtained for a client queue using the CQSQUERY FUNC=QTYPE request with QTYPE=queuetypeaddress specified.

RETCODE=returncodeaddress
Output parameter that specifies the address of a 4-byte field to contain the CQSQUERY return code.

If the return code in register 15 is nonzero, the values in the return and reason code fields are invalid, because the CQS interface detected an error and was unable to send the request to CQS.

RSCODE=reasoncodeaddress
Output parameter that specifies the address of a 4-byte field to contain the CQSQUERY reason code.

STATSZAR=statisticssizeaddress
Output parameter that specifies the address of a 4-byte field to contain the
length of a single statistics entry returned in the output buffer for a CQSQUERY FUNC=STATISTICS request.

If partial data is returned, the size of the required buffer can be obtained by multiplying the value returned in this field by the number of list entries specified.

**Return and Reason Codes for CQSQUERY**

Table 39 lists the return and reason code combinations that can be returned for CQSQUERY requests. Use a CQSQUERY FUNC=DSECT request to include equate statements in your program for the return and reason codes.

**Table 39. CQSQUERY Return and Reason Codes**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000000'</td>
<td>X'00000000'</td>
<td>Request completed successfully.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000120'</td>
<td>The buffer size (buffersize) is less than the query-data size (querydatasize). Partial data is returned. querydatasize points to the actual buffer size needed to contain all the data.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000124'</td>
<td>buffersize is too small to contain data for number of entries specified in list.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000128'</td>
<td>No data objects on queue type.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000010'</td>
<td>cqstoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000012'</td>
<td>connecttoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000014'</td>
<td>FUNC is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000024'</td>
<td>bufferaddress is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000028'</td>
<td>buffersize is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'0000002C'</td>
<td>statisticssize or querydatasize is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000030'</td>
<td>queueage is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000034'</td>
<td>queuetype is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000038'</td>
<td>count is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'0000003C'</td>
<td>listaddress is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000040'</td>
<td>CQSQUERY FUNC=QNAME, CQSQUERY FUNC=QTYPE, or CQSQUERY FUNC=QOBS is not allowed for a resource structure.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000044'</td>
<td>Parmlist version is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000048'</td>
<td>List version is invalid.</td>
</tr>
<tr>
<td>X'0000000C'</td>
<td>X'00000050'</td>
<td>Request completed successfully for at least one, but not all, list entries. See compcode for individual errors.</td>
</tr>
<tr>
<td>X'0000000C'</td>
<td>X'00000054'</td>
<td>Request failed for all list entries. See compcode for individual errors.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000058'</td>
<td>A CQSRSYNC is required for this structure.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'0000005C'</td>
<td>Structure inaccessible. Retry request later.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000060'</td>
<td>No CQS address space.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000064'</td>
<td>CQS internal error.</td>
</tr>
</tbody>
</table>
CQSREAD

CQSREAD Request

Format for CQSREAD

**CONTINUE Function of CQSREAD:** Use the CONTINUE function of a CQSREAD request to retrieve the rest of a data object after partial data is returned for a prior CQSREAD request.

```
CQSREAD—FUNC=CONTINUE—CQSTOKEN=cqstokenaddress

CONTOKEN=connecttokenaddress—PARM=parmaddress—LCKTOKEN=locktokenaddress

BUFFER=bufferaddress—BUFSIZE=buffersize—OBJSIZE=dataobjectsizeaddress

ECB=ecbaddress

RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress
```

**DSECT Function of CQSREAD:** Use the DSECT function of a CQSREAD request to include equate (EQU) statements in your program for the CQSREAD parameter list length, CQSREAD return and reason codes, and literals that can be used to build the OPTWORD1 parameter.

```
CQSREAD—FUNC=DSECT
```

**READ Function of CQSREAD with LOCAL=NO:** Use the CQSREAD request with the LOCAL=NO parameter to retrieve a copy of the client data object from a specific queue and lock it.

```
CQSREAD—FUNC=READ—CQSTOKEN=cqstokenaddress—CONTOKEN=connecttokenaddress

PARM=parmaddress—LCKTOKEN=locktokenaddress—QNAME=queuenameaddress

A

OPTWORD1=optionwordvalue

BUFFER=bufferaddress—BUFSIZE=buffersize

OBJSIZE=dataobjectsizeaddress—UOW=uowaddress

TIMESTAMP=timestampaddress

ECB=ecbaddress

RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress
```

A:

```
QPOS=FIRST
QPOST=LAST
PARTIAL=YES
PARTIAL=NO
LOCAL=NO
```
**READ Function of CQSREAD with LOCAL=YES:** Use the CQSREAD request with the LOCAL=YES parameter to retrieve the lock token of a data object previously stored on the shared queues by a CQSPUT LOCAL=YES request. Using this request ensures that the data object remains locked, even in the event of client failure, structure rebuild, or CQS restart.

```
| CQSREAD—FUNC=READ—CQSTOKEN=cqstokenaddress—CONTOKEN=connecttokenaddress |
| PARM=parmaddress—LCKTOKEN=locktokenaddress—QNAME=queueaddressaddress   |
| UOW=uowaddress—LOCAL=YES                                               |
| OPTWORD1=optionwordvalue—ECB=ecbaddress                                  |
| RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress                     |
```

**REREAD Function of CQSREAD:** Use the REREAD function of a CQSREAD request to re-read a locked data object that was read and locked on a prior CQSREAD FUNC=READ request.

```
| CQSREAD—FUNC=REREAD—CQSTOKEN=cqstokenaddress—CONTOKEN=connecttokenaddress |
| PARM=parmaddress—LCKTOKEN=locktokenaddress—QNAME=queueaddressaddress   |
| BUFFER=bufferaddress—BUFSIZE=buffersize—OBJSIZE=dataobjectsizeaddress |
| ECB=ecbaddress—RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress       |
```

**Usage of CQSREAD**
A CQSREAD request retrieves a copy of the client data object from a specific queue. The data object is not deleted from the queue, but for a CQSREAD FUNC=READ request it is locked, which prevents the data object from being accessed by subsequent CQS requests (except ones using the proper lock token). The data object can be retrieved from the beginning or from the end of the queue. The data object is returned in the client buffer provided for the CQSREAD request.

**Restriction:** The CQSREAD request is not supported for resource structures.

A lock token is returned to the client and identifies the data object. This token must be passed to CQS for any requests that act on the locked data object (for example, CQSDEL, CQSMOVE, CQSREAD, or CQSUNLCK).

If the size of the data object retrieved is greater than the size of the client buffer and PARTIAL=YES is specified, the amount of data that fits in the client buffer is returned to the client. A return or reason code is also returned, indicating a partial data object is returned, as is the actual data object size.

If the size of the data object retrieved is greater than the size of the client buffer and PARTIAL=NO is specified, no data object is returned. A return and reason code is returned, indicating that no data object is returned because the client buffer size is too small. The actual data object size is also returned to the client.

If the size of the data object retrieved is the same size as or smaller than the client buffer, the complete data object is moved into the buffer, and the rest of the buffer is not changed. The data object size is also returned to the client.
A CQSREAD FUNC=CONTINUE request retrieves the rest of the data object when partial data is returned on a prior CQSREAD request.

**Attention:** This request could result in an error after a CQS restart because the current position might be lost across CQS restart.

A CQSREAD FUNC=REREAD request re-reads a locked data object that was previously read and locked (a prior CQSREAD FUNC=READ request). The data object remains locked.

**Related Reading:** See "Example of Using a CQS Request: CQSREAD" on page 165 for an example of how to use a CQSREAD request for a CQS client.

**Parameter Description:**

- **BUFFER=bufferaddress**
  Four-byte input parameter that specifies the address of the client buffer that will hold the data object retrieved from the queue.

- **BUFSIZE=buffersize**
  Four-byte input parameter that specifies the size of the client buffer.

- **CONTOKEN=connecttokenaddress**
  Input parameter that specifies the address of the 16-byte connect token that uniquely identifies the client’s connection to a particular coupling facility structure managed by this CQS. The connect token is returned by the CQSCONN request.

- **CQSTOKEN=cqstokenaddress**
  Input parameter that specifies the address of the 16-byte CQS registration token that uniquely identifies the client’s connection to CQS. The registration token is returned by the CQSREG request.

- **ECB=ecbaddress**
  Four-byte input parameter that specifies the address of the MVS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise it is processed synchronously.

- **LCKTOKEN=locktokenaddress**
  Input and output parameter that specifies the address of the 16-byte lock token for the data object that was locked by the CQSREAD request.

  For a CQSREAD FUNC=READ request, the lock token is zero on input. It is also used as an output area to hold the lock token returned to the client.

  For a CQSREAD FUNC=REREAD or FUNC=CONTINUE request, this field is an input area that contains the lock token returned on a prior CQSREAD request.

- **LOCAL=NO | YES**
  Input parameter that indicates whether or not the client should process a local copy of the data object from the client address space.

  **NO**
  Indicates the client wants CQS to return the data object from the specified client queue and lock the data object. This causes CQS to access the coupling facility to retrieve the data object.

  **YES**
  Indicates that the client is processing a local copy of a data object from its local buffers. This request returns the lock token of the data object.
which the client can use to access the copy of the data object on the shared queues. The data object was placed on the shared queues by a CQSPUT LOCAL=YES request.

By using a local copy of the data object, the client can reduce the performance overhead of using shared queues. As long as the data object is on the shared queues, it can be recovered if the client fails. As long as the data object remains locked, it is not available to any other client.

The data object is not returned to the client on a CQSREAD request because the client has the local copy. If the client does not issue the CQSREAD LOCAL=YES request and the connection between the client and CQS is lost, CQS unlocks the data object and makes it available to any client.

**Restriction:** If you specify LOCAL=YES, you cannot use the TIMESTAMP parameter.

The LOCAL parameter cannot be used when the OPTWORD1 parameter is specified. If the OPTWORD1 parameter is specified instead of LOCAL, you can use the following equate (EQU) statements to generate the value for the OPTWORD1 parameter:

```
CQSREAD_LCLYEQUX LOCAL=YES
CQSREAD_LCLNEQUX LOCAL=NO
```

**OBJSIZE=**dataobjectsizetaddress

Output parameter to receive the address of a 4-byte field that holds the size of the data object. If the data object size is greater than the client buffer size, this field contains the actual data object size. If partial data is returned, the size of the data object returned is the size of the client buffer specified.

**OPTWORD1=**optionwordvalue

Four-byte input parameter that specifies the literals for this request. This parameter can be used instead of LOCAL, PARTIAL, and QPOS. Equate (EQU) statements for the literal values are listed in the descriptions for the LOCAL, PARTIAL, and QPOS parameters. Equate statements can also be generated by using the DSECT function. The OPTWORD1 parameter cannot be used if LOCAL, PARTIAL, or QPOS is specified.

**Requirement:** If you code the OPTWORD1 parameter, you must pass a value that is composed of one equate value for each literal value supported by this macro.

**PARM=**parmaddress

Four-byte input parameter that specifies the address of a parameter list used by the request to pass parameters to CQS. The length of the storage area must be at least equal to the EQU value CQSREAD_PARM_LEN (defined using the FUNC=DSECT request).

**PARTIAL=YES | NO**

Input parameter that specifies whether partial data is to be retrieved, and whether the data object is to be locked if the data object size is greater than the client buffer size.

**YES** If the data object size is greater than the client buffer size, the data object is locked and partial data is returned in the client buffer. The actual size of the data object is returned in dataobjectsizet.

**NO** If the data object size is greater than the client buffer size, the data
object is neither locked nor retrieved. The actual size of the data object is returned in `dataobjectsize`.

The PARTIAL parameter cannot be used when the OPTWORD1 parameter is specified. If the OPTWORD1 parameter is specified instead of PARTIAL, you can use the following equate (EQU) statements to generate the value for the OPTWORD1 parameter:

```plaintext
CQSREAD_PRTLNEQUX PARTIAL=NO
CQSREAD_PRTLYEQUX PARTIAL=YES
```

**QNAME**=`queuenameaddress`

Input parameter that specifies the address of the 16-byte queue name from which the data object is to be retrieved. The first byte of the queue name identifies the queue type.

**QPOS**=`FIRST | LAST`

Input parameter that specifies the position on the queue from which the data object is to be retrieved.

- **FIRST** The data object is retrieved from the beginning of the queue.
- **LAST** The data object is retrieved from the end of the queue.

The QPOS parameter cannot be used when the OPTWORD1 parameter is specified. If the OPTWORD1 parameter is specified instead of QPOS, you can use the following equate (EQU) statements to generate the value for the OPTWORD1 parameter:

```plaintext
CQSREAD_QPOSLEQUX QPOS=LAST
CQSREAD_QPOSFEQUX QPOS=FIRST
```

**RETCODE**=`returncodeaddress`

Output parameter that specifies the address of a 4-byte field to contain the CQSREAD return code.

If the return code in register 15 is nonzero, the values in the return and reason code fields are invalid, because the CQS interface detected an error and was unable to send the request to CQS.

**RSNCODE**=`reasoncodeaddress`

Output parameter that specifies the address of a 4-byte field to contain the CQSREAD reason code.

**TIMESTAMP**=`timestampaddress`

Four-byte output parameter that specifies the address of an eight-byte field to contain the timestamp of when the data object was placed on the queues.

**Attention:** If LOCAL=YES is specified, CQS does not read the data object from the structure, and the timestamp cannot be obtained.

**UOW**=`uowaddress`

Output parameter that specifies the address of a 32-byte area to hold the unit of work (UOW) of the data object retrieved from the queue. The UOW was generated by the client that put the data object on the queue using a CQSPUT request.

### Return and Reason Codes for CQSREAD

Table 40 on page 141 lists the return and reason code combinations that can be returned for CQSREAD requests. Use a CQSREAD FUNC=DSECT request to include equate statements in your program for the return and reason codes.
Table 40. CQSREAD Return and Reason Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000000'</td>
<td>X'00000000'</td>
<td>Request completed successfully.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000120'</td>
<td>The buffer size (buffersize) is less than the data object size (dataobjectsize). Partial data is returned. dataobjectsize contains the address of the actual data object size.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000124'</td>
<td>The buffer size (buffersize) is less than the data object size (dataobjectsize). No data is returned because PARTIAL=NO was specified. dataobjectsize contains the address of the actual data object size.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000128'</td>
<td>No data object to retrieve on queue name specified.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'0000012C'</td>
<td>No partial data to return.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000210'</td>
<td>cqstoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000214'</td>
<td>connecttoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000218'</td>
<td>FUNC is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'0000021C'</td>
<td>locktoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000220'</td>
<td>queuename is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000224'</td>
<td>bufferaddress is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000228'</td>
<td>buffersize is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'0000022C'</td>
<td>dataobjectsize is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000230'</td>
<td>uow is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000234'</td>
<td>Lock token address is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000238'</td>
<td>The request specified LOCAL=YES, but the requested object was placed on the structure using LOCAL=NO.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'0000027C'</td>
<td>CQSREAD is not allowed for a resource structure.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000284'</td>
<td>Parmlist version is invalid.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000400'</td>
<td>A CQSRSYNC is required for this structure.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000404'</td>
<td>Structure inaccessible. Retry request later.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000408'</td>
<td>Current position lost; cannot process CQSREAD FUNC=CONTINUE request.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000410'</td>
<td>No CQS address space.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000440'</td>
<td>Object lost because of rebuild.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000500'</td>
<td>CQS internal error.</td>
</tr>
</tbody>
</table>

CQSRECVR Request

Format for CQSRECVR

DELETE Function of CQSRECVR: Use the DELETE function of a CQSRECVR request to delete one data object associated with a UOW from the cold queue.
CQSRECVR

DSECT Function of CQSRECVR: Use the DSECT function of a CQSRECVR request to include equate (EQU) statements in your program for the CQSRECVR parameter list length, CQSRECVR return and reason codes, and literals that can be used to build the OPTWORD1 parameter.

RETRIEVE Function of CQSRECVR: Use the RETRIEVE function of a CQSRECVR request to retrieve a copy of a data object associated with a UOW from the cold queue.

UNLOCK Function of CQSRECVR: Use the UNLOCK function of a CQSRECVR request to unlock a data object associated with a UOW on the cold queue.
Usage of CQSRECVR

The CQSRECVR request allows a client to recover locked data objects that were moved to the CQS cold queue (a CQS private queue) because CQS or the client was cold started.

Restriction: The CQSRECVR request is not supported for resource structures.

A CQSRECVR FUNC=DELETE request deletes a data object associated with a UOW from the cold queue. Only one data object is deleted.

A CQSRECVR FUNC=RETRIEVE request retrieves a copy of the data object associated with a UOW from the cold queue. The data object remains on the cold queue, and is available for other CQSRECVR requests. The data object is returned in the client buffer specified for the CQSRECVR FUNC=RETRIEVE request.

If the data object is the same size as or smaller than the client buffer provided, the data object is returned in the buffer, and the rest of the buffer is not changed. The size of the data object is returned to the client.

If the size of the data object is greater than the size of the client buffer, the data object is not returned. The size of the data object is returned to the client.

A CQSRECVR FUNC=UNLOCK request unlocks a data object associated with a UOW on the cold queue. The data object is moved from the cold queue to the original client queue, and is available for other CQS requests. The position to which the data object should be moved can be specified by the client.

Parameter Description:

**BUFFER=bufferaddress**

Four-byte input parameter that specifies the address of the client buffer that will hold the data object retrieved from the queue.
BUFSIZE=buffersize
Four-byte input parameter that specifies the size of the client buffer.

CLDTOKEN=coldqueuetokenaddress
Input parameter that specifies the address of a 16-byte cold-queue token, which along with the UOW identifies the data object that is to be recovered from the CQS cold queue (COLDQ).

The cold-queue token is passed to the client in the SEVX_RETOKEN field of the Resync entry in the CQS Structure Event exit routine. This exit routine is called for a CQS-initiated resynchronization when the UOW status is COLD.

CONTOKEN=connecttokenaddress
Input parameter that specifies the address of a 16-byte connect token that uniquely identifies the client's connection to a particular coupling facility structure managed by this CQS. The connect token is returned by the CQSCONN request.

CQSTOKEN=cqstokenaddress
Input parameter that specifies address of the 16-byte CQS registration token that uniquely identifies the client's connection to CQS. The registration token is returned by the CQSREG request.

ECB=ecbaddress
Four-byte input parameter that specifies the address of the MVS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise it is processed synchronously.

OBJSIZE=dataobjectsizaddress
Output parameter that specifies the address of a 4-byte area to hold the size of the data object. If the data object size is greater than the client buffer size, this field contains the actual data object size. If partial data is returned, the data object returned is the size of the client buffer specified.

OPTWORD1=optionwordvalue
Four-byte input parameter that specifies the literals for this request. This parameter can be used instead of QPOS. Equate (EQU) statements for the literal values are listed in the description of the QPOS parameter. Equate statements can also be generated by using the DSECT function. The OPTWORD1 parameter cannot be used if QPOS is specified.

Requirement: If you code the OPTWORD1 parameter, you must pass a value that is composed of one equate value for each literal value supported by this macro.

PARM=parmaddress
Four-byte input parameter that specifies the address of a parameter list used by the request to pass parameters to CQS. The length of the storage area must be at least equal to the EQU value CQSRECVR_PARM_LEN (defined using the FUNC=DSECT request).

QPOS=SYSTEM | FIRST | LAST
Input parameter that specifies the position on the queue to which the unlocked data object is to be added. The default is SYSTEM.

SYSTEM
Indicates the data object is unlocked and added to either the beginning or the end of the queue, depending on its original position. If the CQSREAD request that locked this data object
obtained the data object from the beginning of the queue, the data object is unlocked and added to the beginning of the queue. If the CQSREAD request obtained the data object from the end of the queue, the data object is unlocked and added to the end of the queue.

**FIRST** Indicates the data object is unlocked and added to the beginning of the queue.

**LAST** Indicates the data object is unlocked and added to the end of the queue.

The QPOS parameter cannot be used when the OPTWORD1 parameter is specified. If the OPTWORD1 parameter is specified instead of QPOS, you can use the following equate (EQU) statements to generate the value for the OPTWORD1 parameter:

CQSRECVR_QPOSSEQUX QPOS=SYSTEM
CQSRECVR_QPOSSEQUX QPOS=FIRST
CQSRECVR_QPOSSEQUX QPOS=LAST

**RETCODE=returncodeaddress**
Output parameter that specifies the address of a 4-byte field to contain the CQSRECVR return code.

If the return code in register 15 is nonzero, the values in the return and reason code fields are invalid, because the CQS interface detected an error and was unable to send the request to CQS.

**RSNCODE=reasoncodeaddress**
Output parameter that specifies the address of a 4-byte field to contain the CQSRECVR reason code.

**UOW=uowaddress**
Input parameter that specifies the address of a 32-byte area to hold the unit of work (UOW) of a data object. The UOW, together with the coldqueuetoken, identifies the data object to be recovered from the cold queue.

The UOW is passed to the client in the SEVX_REUOW field of the Resync entry in the CQS Structure Event exit routine. This exit routine is called for a CQS-initiated resynchronization when the UOW status is COLD.

### Return and Reason Codes for CQSRECVR

Table 41 lists the return and reason code combinations that can be returned for CQSRECVR requests. Use a CQSRECVR FUNC=DSECT request to include equate statements in your program for the return and reason codes.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000000'</td>
<td>X'00000000'</td>
<td>Request completed successfully.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000124'</td>
<td>buffersize is too small.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000128'</td>
<td>Data object for UOW not found on cold queue.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000210'</td>
<td>cqstoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000214'</td>
<td>connecttoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000218'</td>
<td>FUNC is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000224'</td>
<td>bufferaddress is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000228'</td>
<td>buffersize is invalid.</td>
</tr>
</tbody>
</table>
Table 41. CQSRECVR Return and Reason Codes (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000008'</td>
<td>X'0000022C'</td>
<td>dataobjectsize is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000230'</td>
<td>uow is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000234'</td>
<td>coldqueuetoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'0000027C'</td>
<td>CQSRECVR is not allowed for a resource structure.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000284'</td>
<td>Parmlist version is invalid.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000400'</td>
<td>A CQSRSYNC is required for this structure.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000404'</td>
<td>Structure is inaccessible. Retry request later.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000414'</td>
<td>Unable to unlock the data object because the original queue is full. No more data objects can be moved to this queue. CQSRECVR FUNC=UNLOCK requests for other queues are allowed.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000430'</td>
<td>No CQS address space.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000500'</td>
<td>CQS internal error.</td>
</tr>
</tbody>
</table>

CQSREG Request

Format for CQSREG

**DSECT Function of CQSREG:** Use the DSECT function of a CQSREG request to include equate (EQU) statements in your program for the CQSREG parameter list length and CQSREG return and reason codes.

```c
//CQSREG—FUNC=DSECT
```

**REGISTER Function of CQSREG:** Use the REGISTER function of a CQSREG request to register a client with a CQS.

```c
//CQSREG—FUNC=REGISTER—PARM=parmaddress—CQSSSN=cqssubsystemnameaddress
```

Usage of CQSREG

A CQSREG request registers a client to CQS. If the registration is successful, a CQS token is returned. This token represents the client’s registration with CQS and must be used with all subsequent CQS requests to identify the client.

A CQSREG FUNC=REGISTER request must be the first CQS request a client makes. Also, after a CQS abnormal termination and restart, a CQSREG FUNC=REGISTER request is required before the client can resume issuing CQS requests.

Parameter Description:
CLIENT=clientnameaddress
Input parameter that specifies the address of the 8-byte name of the client registering to CQS. The client name must be unique among all clients that are registered to the same CQS and to all the CQSs that are sharing the same queues.

CQSTOKEN=cqstokenaddress
Output parameter that specifies the address of a 16-byte area to receive the CQS registration token that uniquely identifies the client's connection to CQS. The registration token is returned by a successful CQSREG request.

CQSSSN=cqssubsystemnameaddress
Input parameter that specifies the address of the 4-byte subsystem name of the CQS to which the client would like to connect. This parameter should match the SSN= parameter of the CQSIPxxx PROCLIB member for the CQS to which the client would like to connect.

EVENT=cqseventexit
Four-byte input parameter that specifies the CQS Event exit routine address.

EVENTPARM=eventparmaddress
Input parameter that specifies the address of a 4-byte area that contains client data that CQS passes to the CQS Event exit routine every time the exit is called.

PARM=parmaddress
Four-byte input parameter that specifies the address of a parameter list used by the request to pass parameters to CQS. The length of the storage area must be at least equal to the EQU value CQSREG_PARM_LEN (defined using the FUNC=DSECT request).

RETCODE=returncodeaddress
Output parameter that specifies the address of a 4-byte field to contain the CQSREG return code. The CQSREG return code is returned both in this field and in register 15.

RSNCODE=reasoncodeaddress
Output parameter that specifies the address of a 4-byte field to contain the CQSREG reason code. The CQSREG reason code is returned both in this field and in register 0.

VERSION=cqsversionaddress
Output parameter that specifies the address of a 4-byte area to receive the CQS version number. The version number has the following format:

00vvrrmm.

00 This byte is reserved for future use. Currently, it is always 00.
vv Version number.
rr Release number.
mm Modification level or sub-release number.

Example: CQS version 1.1.0 is shown as X'00010100'.
### Table 42. CQSREG Return and Reason Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000000'</td>
<td>X'00000000'</td>
<td>Request completed successfully.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000100'</td>
<td>Client is already registered to CQS.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000244'</td>
<td>Clientname is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000248'</td>
<td>The CQSREG parameter list version is invalid. This error is probably caused by a difference in versions between the CQS client and the CQS address space the client is trying to use.</td>
</tr>
<tr>
<td>X'00000100'</td>
<td>X'0000040C'</td>
<td>CQS shutdown is pending.</td>
</tr>
<tr>
<td>X'00000100'</td>
<td>X'00000430'</td>
<td>The CQS address space is not active. The CQS address space must be started.</td>
</tr>
<tr>
<td>X'00000100'</td>
<td>X'00000438'</td>
<td>Another address space is already registered with CQS using the client ID (passed on a CQSREG request).</td>
</tr>
<tr>
<td>X'00000100'</td>
<td>X'00000440'</td>
<td>The user ID of the client address space is not authorized to register with this CQS.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000500'</td>
<td>CQS internal error.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000504'</td>
<td>Unable to obtain storage in client’s address space for CQS’s use.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000508'</td>
<td>Unable to obtain storage (CCIB).</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'0000050C'</td>
<td>Unable to obtain storage (CRET).</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000510'</td>
<td>CQS internal error (Loc ASCB).</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000514'</td>
<td>Unable to establish MVS Resource Manager routine to monitor CQS for the registering client.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000518'</td>
<td>CQS internal error (ESTAE add).</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'0000051C'</td>
<td>CQS internal error (NmTktn Retrv).</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000520'</td>
<td>CQS internal error (CGCT error).</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000524'</td>
<td>CQS internal error (TTKn error).</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000528'</td>
<td>CQS internal error (ALESERV error).</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'0000052C'</td>
<td>CQS internal error (BPESVC error).</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000530'</td>
<td>Unable to establish MVS Resource Manager routine to monitor the client for CQS.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000534'</td>
<td>An abend occurred during CQSREG processing.</td>
</tr>
</tbody>
</table>

### CQSRSYNC Request

#### Format for CQSRSYNC

**DSECT Function of CQSRSYNC:** Use the DSECT function of a CQSRSYNC request to include equate (EQU) statements in your program for the CQSRSYNC parameter list length and CQSRSYNC return and reason codes.
**RSYNCCOLD Function of CQSRSYNC:** Use the RSYNCCOLD function of a CQSRSYNC request when the client is performing a cold start and does not have information on unresolved UOWs.

```
CQSRSYNC--FUNC=RSYNCCOLD--CQSTOKEN=cqstokenaddress

(CONTOKEN=connecttokenaddress--PARM=parmaddress)

(RETCODE=returncodeaddress--RSNCODE=reasoncodeaddress)

LISTVER=1

LISTVER=listversion
```

**RSYNCWARM Function of CQSRSYNC:** Use the RSYNCWARM function of a CQSRSYNC request when the client is performing a warm or emergency restart and has information on unresolved UOWs that need to be resolved with CQS.

```
CQSRSYNC--FUNC=RSYNCWARM--CQSTOKEN=cqstokenaddress

(CONTOKEN=connecttokenaddress--PARM=parmaddress--COUNT=count)

(RETCODE=returncodeaddress)

LIST=listaddress

(ESCB=ecbaddress)

LISTVER=1

LISTVER=listversion

RSNCODE=reasoncodeaddress
```

**Usage of CQSRSYNC**

A CQSRSYNC request allows a client to resynchronize indoubt data for one structure with CQS. This request must be the first request the client issues following a CQSCONN request.

**Restriction:** The CQSRSYNC request is not supported for resource structures.

A CQSRSYNC request is required even if the client does not have any indoubt units of work (UOWs) to resolve, for example when the client performs a cold start or a warm start after a normal termination. This request is required because CQS might have information about a connection and have unresolved UOWs to process.

If there are unresolved UOWs, CQS calls the client’s Structure Event exit routine as part of resynchronization. CQS calls the routine to inform the client of UOWs that CQS knows about and that the client did not pass on the CQSRSYNC request. This process is referred to as CQS-initiated resynchronization.

The exit routine is called during client cold start or restart only if CQS has unresolved UOWs. The Structure Event exit routine can be called more than once.
for CQS-initiated resynchronization. For each UOW passed to the exit routine, the client is responsible for taking the correct action to resolve the UOW based on the status returned by CQS.

If CQS cold started, CQS has no knowledge of client UOWs. In this case, the resynchronization list is not processed. CQS looks for CQSREAD requests that were incomplete at the time CQS terminated. If there is incomplete work, the data objects are moved to the cold queue and the Structure Event exit routine is called to inform the client of the unresolved UOWs for the data objects.

After the CQSRSYNC request completes, some UOWs might have a deferred resynchronization status. This status indicates that CQS is still resynchronizing the UOW. When CQS completes resynchronization, the Structure Event exit routine is called to indicate the state of the UOW. Deferred resynchronization only applies to UOWs that CQS cannot resynchronize during the CQSRSYNC request, and does not occur for a client cold start. The exit routine is called once for each deferred UOW, and so the exit routine can be called multiple times for deferred resynchronization.

Parameter Description:

**CONTOKEN=**<connecttokenaddress>
Input parameter that specifies the address of the 16-byte connect token that uniquely identifies the client’s connection to a particular coupling facility structure managed by this CQS. The connect token is returned by the CQSCONN request.

**COUNT=**<count>
Four-byte input parameter that specifies the number of entries in the resync list.

**CQSTOKEN=**<cqstokenaddress>
Input parameter that specifies the address of the 16-byte CQS registration token that uniquely identifies the client’s connection to CQS. The registration token is returned by the CQSREG request.

**ECB=**<ecbaddress>
Four-byte input parameter that specifies the address of the MVS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise it is processed synchronously.

**LIST=**<listaddress>
Four-byte input parameter that specifies the address of the resync list. Each entry contains an indoubt UOW that the client needs to resolve. Some fields in each entry must be initialized by the client prior to the CQSRSYNC request. Other fields are returned by CQS upon completion of the CQSRSYNC request.

The CQSRSYNL list entry DSECT maps the list entries and can be used by the client. Multiple list entries must reside in contiguous storage.

Each list entry contains the following:

- **clientdata**
  Four-byte input parameter that specifies the client data field. This parameter is optional. CQS does not use data stored in this entry.

- **uow**
  Thirty-two-byte input parameter that specifies the unit of work
identifier for the queue. This parameter is required and must be initialized by the client prior to the CQSRSYNC request.

**clientstatus**
Two-byte input parameter that contains the status of the UOW. This status represents the last action the client performed for this UOW. This parameter is required and must be initialized by the client prior to the CQSRSYNC request.

Possible values for the status are shown in Table 43.

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0010'</td>
<td>Put Complete</td>
</tr>
<tr>
<td></td>
<td>The last (or only) CQSPUT request in a series of CQSPUT requests has been issued for the UOW. All data objects for the UOW are assumed to be on the coupling facility.</td>
</tr>
<tr>
<td>X'0020'</td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td>The data object for the UOW is assumed to be locked on the coupling facility.</td>
</tr>
<tr>
<td>X'0030'</td>
<td>Unlock</td>
</tr>
<tr>
<td></td>
<td>A CQSUNLCK request with lock token was issued for the UOW. The data object is assumed to have been unlocked and made available on the work queue on the coupling facility.</td>
</tr>
<tr>
<td>X'0040'</td>
<td>Move</td>
</tr>
<tr>
<td></td>
<td>A CQSMOVE request with lock token was issued for the UOW. The data object is assumed to have been moved to a new queue on the coupling facility.</td>
</tr>
<tr>
<td>X'0050'</td>
<td>Delete</td>
</tr>
<tr>
<td></td>
<td>A CQSDEL request with lock token was issued for the UOW. The data object is assumed to have been deleted from the coupling facility.</td>
</tr>
</tbody>
</table>

**cqsstate**
Two-byte output parameter to receive the resulting state of the UOW from CQS. This parameter is returned by CQS as a result of the CQSRSYNC request.

Possible values for the status are shown in Table 44.

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0010'</td>
<td>Put Insync</td>
</tr>
<tr>
<td></td>
<td>Client status is Put Complete. CQS status is Put Complete. CQS knows about the UOW and all data objects for the UOW are out on the coupling facility. A put token is returned for the UOW. The client should use the put token to issue a CQSPUT FUNC=FORGET request.</td>
</tr>
<tr>
<td>Status</td>
<td>Meaning</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>X'0012'</td>
<td>Resync Deferred</td>
</tr>
<tr>
<td></td>
<td>Client status is Put Complete. CQS status is Indoubt. This status is only returned for recoverable UOWs. CQS knows about the UOW but is still in the process of determining its status. The client should wait until its Structure Event exit routine is called by CQS. CQS will post the client's Structure Event exit routine, passing the UOW and a status for the UOW. If the status is PUT Insync, a put token for the UOW is also returned. The client should use the put token to issue a CQSPUT FUNC=FORGET request. If the status is PUT Failed, the client must reissue the CQSPUT FUNC=PUT request. If the status is Unknown, the data object might or might not be on the coupling facility.</td>
</tr>
<tr>
<td>X'0020'</td>
<td>Read Insync</td>
</tr>
<tr>
<td></td>
<td>Client status is Read. CQS status is Read Complete. CQS found the data object for the UOW to be locked. A lock token is returned for the UOW. The client should use this lock token on subsequent CQS requests for the data object with this UOW.</td>
</tr>
<tr>
<td>X'0030'</td>
<td>Unlock Insync</td>
</tr>
<tr>
<td></td>
<td>Client status is Read Unlock. CQS status is Unlock Insync. CQS found the data object for the UOW to be locked, and unlocked it. No further action is required by the client.</td>
</tr>
<tr>
<td>X'0050'</td>
<td>Delete Insync</td>
</tr>
<tr>
<td></td>
<td>Client status is Delete. CQS status is Delete Insync. CQS found the data object for the UOW to be locked and deleted it. No further action is required by the client.</td>
</tr>
<tr>
<td>X'00F1'</td>
<td>Locked</td>
</tr>
<tr>
<td></td>
<td>One of the following conditions exists:</td>
</tr>
<tr>
<td></td>
<td>• Client status is Delete. CQS status is Locked. CQS found the UOW to be locked, but could not delete the data object from the structure. The data object remains locked. A lock token is returned for the UOW. The client should use this lock token and reissue the CQSDEL request.</td>
</tr>
<tr>
<td></td>
<td>• Client status is Move. CQS status is Locked. CQS found the data object for UOW in Locked state. The CQSMOVE could not be completed because the new queue name is not available. A lock token is returned for the UOW. The client should use this lock token and reissue the CQSMOVE request.</td>
</tr>
<tr>
<td></td>
<td>• Client status is Unlock. CQS status is Locked. CQS found the UOW to be locked, but could not unlock the data object. The data object remains locked. A lock token is returned for the UOW. The client should use this lock token and reissue the CQSUNLCK request.</td>
</tr>
<tr>
<td>X'00F2'</td>
<td>Unknown</td>
</tr>
<tr>
<td></td>
<td>Client status is any valid client status. The UOW is unknown to CQS.</td>
</tr>
<tr>
<td></td>
<td>If the client believes the UOW to be in PUT Complete status, the client must determine whether or not to reissue the CQSPUT request. If the client believes the UOW to have a status of Delete, Move, Read, or Unlock, the prior request could have completed.</td>
</tr>
</tbody>
</table>

**resynctoken**

Sixteen-byte output parameter to receive a token that the client
uses to complete processing for the UOW. When the state is Put
Insync, this field contains the put token. When the state is Locked,
this field contains the lock token. This field is returned by CQS as a
result of the CQSRSYNC request.

compcode
Four-byte output field to receive the completion code from the
request. Possible completion codes are:

X'00000000'  CQS successfully processed this UOW. Client and
CQS are in sync for this UOW. An Insync state is
returned for this UOW.

X'00000004'  CQS successfully processed this UOW. Client and
CQS are not in sync for this UOW. CQS returns its
known state for this UOW.

X'00000008'  clientstatus is invalid. CQS could not resynchronize
this UOW. The cqsstate is not returned.

X'0000000C'  uow is invalid. CQS could not resynchronize this
UOW. The cqsstate is not returned.

X'00000010'  CQS internal error. CQS could not resynchronize
this UOW. The cqsstate is not returned.

LISTVER=1 | listversion
Input parameter that specifies an equate for the list version. Use the
DSECT function of a CQSRSYNC request to include equate (EQU)
statements in your program for the CQSRSYNC list versions.

PARM=parmaddress
Four-byte input parameter that specifies the address of a parameter list
used by the request to pass parameters to CQS. The length of the storage
area must be at least equal to the EQU value CQSRSYNC_PARM_LEN
(defined using the FUNC=DSECT request).

RETCODE=returncodeaddress
Output parameter that specifies the address of a 4-byte field to contain the
CQSRSYNC return code.

If the return code in register 15 is nonzero, the values in the return and
reason code fields are invalid, because the CQS interface detected an error
and was unable to send the request to CQS.

RSNCODE=reasoncodeaddress
Output parameter that specifies the address of a 4-byte field to contain the
CQSRSYNC reason code.

Return and Reason Codes for CQSRSYNC
Table 45 lists the return and reason code combinations that can be returned for
CQSRSYNC requests. Use a CQSRSYNC FUNC=DSECT request to include
equate statements in your program for the return and reason codes.

Table 45. CQSRSYNC Return and Reason Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000000'</td>
<td>X'00000000'</td>
<td>Request completed successfully and all list entries are in sync. The</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Structure Event exit routine is called for CQS resync. The client can</td>
</tr>
<tr>
<td></td>
<td></td>
<td>now issue CQS requests to write or retrieve data for this structure.</td>
</tr>
</tbody>
</table>
### Table 45. CQRSYNC Return and Reason Codes (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000004'</td>
<td>X'00000110'</td>
<td>CQS was cold started. No list entries were processed. CQS did not find any unresolved UOWs. The Structure Event exit routine is not called. The client can now issue CQS requests to write or retrieve data for this structure.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000114'</td>
<td>Client was cold started. CQS did not find any unresolved UOWs. The Structure Event exit routine is not called. The client can now issue CQS requests to write or retrieve data for this structure.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'00000118'</td>
<td>CQS was cold started. No list entries were processed. CQS did find some unresolved UOWs and marked them as being in cold status. The Structure Event exit routine is called to inform the client of the unresolved UOWs. The client can now issue CQS requests to write or retrieve data for this structure.</td>
</tr>
<tr>
<td>X'00000004'</td>
<td>X'0000011C'</td>
<td>Client was cold started. CQS did find some unresolved UOWs. The Structure Event exit routine is called to inform the client of the unresolved UOWs. The client can now issue CQS requests to write or retrieve data for this structure.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000210'</td>
<td>cqstoken is invalid. No list entries were processed. The Structure Event exit routine is not called. The client must reissue the CQRSYNC request.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000214'</td>
<td>connecttoken is invalid. No list entries were processed. The Structure Event exit routine is not called. The client must reissue the CQRSYNC request.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000218'</td>
<td>FUNC is invalid. The client must reissue the CQRSYNC request.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000254'</td>
<td>listaddress is invalid. No list entries were processed. The Structure Event exit routine is not called. The client must reissue the CQRSYNC request.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'0000027C'</td>
<td>CQRSYNC is not allowed for a resource structure.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000284'</td>
<td>Parmlist version is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000288'</td>
<td>List version is invalid.</td>
</tr>
<tr>
<td>X'0000000C'</td>
<td>X'00000300'</td>
<td>Request succeeded for at least one, but not all, list entries. At least one list entry is in sync. See compcode in each list entry for individual errors. The Structure Event exit routine is called for CQS resync. The client can now issue CQS requests to write or retrieve data for this structure.</td>
</tr>
<tr>
<td>X'0000000C'</td>
<td>X'00000304'</td>
<td>Request failed for all list entries. None of the list entries are in sync. See compcode in each list entry for individual errors. The Structure Event exit routine is called for CQS resync. The client can now issue CQS requests to write or retrieve data for this structure.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000430'</td>
<td>No CQS address space.</td>
</tr>
<tr>
<td>X'00000014'</td>
<td>X'00000500'</td>
<td>CQS internal error.</td>
</tr>
</tbody>
</table>
CQSSHUT Request

Format for CQSSHUT

DSECT Function of CQSSHUT: Use the DSECT function of a CQSSHUT request to include equate (EQU) statements in your program for the CQSSHUT parameter list length and CQSSHUT return and reason codes.

```
CQSSHUT—FUNC=DSECT
```

QUIESCE Function of CQSSHUT: Use the QUIESCE function of a CQSSHUT request to terminate CQS.

```
CQSSHUT—FUNC=QUIESCE—CQSTOKEN=cqstokenaddress—PARM=parmaddress
```

Usage of CQSSHUT

A CQSSHUT request notifies CQS to terminate after all clients have disconnected. After the CQSSHUT request is issued, CQS stops accepting CQSCONN requests. CQS continues to accept input or output requests, so that clients can complete work in progress. In order to complete the shutdown process, clients must stop working and issue CQSDISC requests to disconnect from CQS. After all clients have disconnected, CQS terminates all tasks and returns control to MVS.

Parameter Description:

CQSTOKEN= cqstokenaddress

Input parameter that specifies the address of the 16-byte CQS registration token that uniquely identifies the client’s connection to CQS. The registration token is returned by the CQSREG request.

ECB= ecbaddress

Four-byte input parameter that specifies the address of the MVS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise, it is processed synchronously.

PARM= parmaddress

Four-byte input parameter that specifies the address of a parameter list used by the request to pass parameters to CQS. The length of the storage area must be at least equal to the EQU value CQSSHUT_PARM_LEN (defined using the FUNC=DSECT request).

RETCODE= returncodeaddress

Output parameter that specifies the address of a 4-byte field to contain the CQSSHUT return code.

If the return code in register 15 is nonzero, the values in the return and reason code fields are invalid, because the CQS interface detected an error and was unable to send the request to CQS.

RSNCODE= reasoncodeaddress

Four-byte output parameter that specifies the address of a field to contain the CQSSHUT reason code.
Return and Reason Codes for CQSSHUT

Table 46 lists the return and reason code combinations that can be returned for CQSSHUT requests. Use a CQSSHUT FUNC=DSECT request to include equate statements in your program for the return and reason codes.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X’00000000’</td>
<td>X’00000000’</td>
<td>Request completed successfully.</td>
</tr>
<tr>
<td>X’00000008’</td>
<td>X’00000210’</td>
<td>cqstoken is invalid.</td>
</tr>
<tr>
<td>X’00000008’</td>
<td>X’00000218’</td>
<td>FUNC is invalid.</td>
</tr>
<tr>
<td>X’00000008’</td>
<td>X’00000284’</td>
<td>Parmlist version is invalid.</td>
</tr>
<tr>
<td>X’00000010’</td>
<td>X’00000444’</td>
<td>CQS initialization is in progress. Reissue the CQSSHUT request after initialization is complete.</td>
</tr>
</tbody>
</table>

CQSUNLCK Request

Format for CQSUNLCK

**DSECT Function of CQSUNLCK:** Use the DSECT function of a CQSUNLCK request to include equate (EQU) statements in your program for the CQSUNLCK parameter list length and CQSUNLCK return and reason codes.

```
CQSUNLCK FUNC=DSECT
```

**UNLOCK Function of CQSUNLCK:** Use the UNLOCK function of a CQSUNLCK request to unlock one or more data objects and move them to the end or beginning of the queue.

```
CQSUNLCK FUNC=UNLOCK CQSTOKEN=cqstokenaddress CONTOKEN=connecttokenaddress PARM=parmaddress COUNT=count LIST=listaddress ECB=ecbaddress RETCODE=returncodeaddress
```

**FORCE Function of CQSUNLCK:** Use the FORCE function of a CQSUNLCK request to forcibly unlock data objects read from the specified queue type by the specified failed CQS client and clean up CQS’s knowledge of the data objects.
Usage of CQSUNLCK
Restriction: The CQSUNLCK request is not supported for resource structures.

A CQSUNLCK FUNC=UNLOCK request unlocks one or more data objects and moves them into the first or last position on the queue. The client passes an unlock list that contains one or more list entries, where each entry is a separate unlock request. A successful CQSUNLCK request invalidates the lock token and makes the data object available to any client for a CQSBRWSE, CQSDEL, CQSMOVE, or CQSREAD request.

The CQSUNLCK FUNC=FORCE request enables a CQS client to forcibly unlock data objects read from the specified queue type by the specified failed CQS client, so that the data objects don’t remain on the LOCKQ until the failed CQS client restarts. Force unlock also removes the CQS’s knowledge of locked data objects, if this CQS processed the CQSREAD requests that locked the data objects.

When a CQS client fails, its locked data objects remain on the LOCKQ until the CQS client restarts, resynchronizes with CQS, and decides what to do with the locked data objects, or until a CQS client forcibly unlocks the data objects. Locked data objects are not accessible by other CQS clients.

Attention: CQS clients should use the CQSUNLCK FUNC=FORCE request with caution. The CQS clients in an IMSplex must apply the following force unlock rules consistently. If not used consistently, the CQSRSYNC request might fail, data objects might remain on the lock queue, read tables might remain in CQS, or data objects might be moved to the COLDQ. When using CQSUNLCK FUNC=FORCE, apply the following rules:

- Define IMSplex with CSL.
  The IMSplex must be defined with a Common Service Layer, so that CQS clients are notified when a CQS client fails.

- Select queue type candidates.
  Select one or more queue types whose data objects are candidates to be forcibly unlocked. All of the data objects with the specified queue type are candidates. There is no way to select specific data objects of a queue type to be forcibly unlocked.

- Forcibly unlock another CQS client’s data objects when CQS client fails.
  When a CQS client fails, it may leave locked data objects on the LOCKQ. Another CQS client should issue the CQSUNLCK FUNC=FORCE request, so that data objects don’t remain on the LOCKQ until the failed CQS client restarts.
CQSUNLCK

Issue a CQSUNLCK FUNC=FORCE request only to forcibly unlock data objects of a CQS client that is currently not active. It is up to the CQS client issuing the CQSUNLCK FUNC=FORCE request to ensure that the target CQS client is not active.

It is up to the CQS clients in the IMSplex to ensure that only one CQS client issues the CQSUNLCK FUNC=FORCE request. All members in an IMSplex defined with a CSL are notified when a member fails. Multiple CQSUNLCK FUNC=FORCE requests may have the following undesirable results:

- Unnecessary CF accesses.
  
  The CQSUNLCK FUNC=FORCE request incurs multiple CF accesses to look at data objects on the candidate queue type. If multiple CQSUNLCK FUNC=FORCE requests are issued, each request makes the same numerous CF accesses. These extra CF accesses are unnecessary and incur additional performance overhead. If the performance overhead of unnecessary CF accesses is unacceptable, it is up to the CQS clients in the IMSplex to ensure that only one CQS client issues the CQSUNLCK FUNC=FORCE.

- Data objects incorrectly unlocked.
  
  If a failed CQS client initializes right away, it might forcibly unlock its own data objects, resync with CQS, and put new data objects on the queue structure, before another CQS client attempts to forcibly unlock the failed CQS client’s data objects. The other CQS client could incorrectly unlock data objects for UOWs that are in flight. It is up to the CQS clients in the IMSplex to ensure that exactly one CQS client forcibly unlocks data objects for the specified client.

- Forcibly unlock CQS client’s own data objects when CQS client initializes.
  
  When a CQS client initializes, it should forcibly unlock its own data objects before issuing CQSRSYNC. This ensures that the CQS client’s data objects are unlocked before resync, in case no other CQS client was available at failure time to do the force unlock. Force unlock also cleans up CQS’s knowledge of the IMS client’s locked data objects, since this CQS processed the CQSREAD request that locked the data objects.

- Resync with CQS, handling UOW’s that are candidates for unlock force.
  
  When building the resync list to pass to CQS on the CQSRSYNC request, mark all candidates for the UNLOCK FORCE with a CQS client status of forced. CQS resync checks for the client status of forced and sets the UOWs to a CQS status of unlock in sync.

- Forcibly unlock other failed CQS clients’ data objects when CQS client initializes.
  
  When a CQS client initializes, it should forcibly unlock the data objects of failed CQS clients, in case no other CQS client was available to do the force unlock when the CQS clients failed. After an initializing CQS client resynchronizes with CQS, it should issue one CQSUNLCK FUNC=FORCE request per failed CQS client, to forcibly unlock data objects on the candidate queue types.

Parameter Description:

CLIENT=clientnameaddress

Eight-byte input field that specifies the CQS client for which to forcibly unlock
data objects. The client name is the same name specified on the CQSREG request when the client registered to CQS. A CQS client can forcibly unlock its own locked data objects before issuing the CQSRSYNC request. A CQS client can forcibly unlock another CQS client’s locked data objects after issuing the CQSRSYNC request.

**CONTOKEN=**connecttokenaddress
Input parameter that specifies the address of the 16-byte connect token that uniquely identifies the client’s connection to a particular coupling facility structure managed by this CQS. The connect token is returned by the CQSCONN request.

**COUNT=**count
Four-byte input parameter that specifies the number of list entries in the unlock list or four-byte output parameter to receive the count of data objects that were forcibly unlocked.

**CQSTOKEN=**cqstokenaddress
Input parameter that specifies the address of the 16-byte CQS registration token that uniquely identifies the client’s connection to CQS. The registration token is returned by the CQSREG request.

**ECB=**ecbaddress
Four-byte input parameter that specifies the address of the MVS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise it is processed synchronously.

**LIST=**listaddress
Four-byte input parameter that specifies the address of the unlock list. Each entry is a separate CQSUNLCK request. Some fields in each entry must be initialized by the client prior to the CQSUNLCK request. Other fields are returned by CQS upon completion of the CQSUNLCK request.

The CQSUNLL list entry DSECT maps the list entries and can be used by the client. Multiple list entries must reside in contiguous storage.

Each list entry contains the following:

**compcode**
Four-byte output field to receive the completion code from the request. Possible completion codes are:

- X'00000000' Request completed successfully.
- X'00000004' locktoken is invalid.
- X'00000008' Structure inaccessible.
- X'0000000C'
  Unable to unlock the data object, because the original queue for the data object is full. No data objects can be moved to the named queue, but CQSUNLCK requests for other queues are allowed.
- X'00000010' CQS internal error
- X'00000014'
  Data object was lost because the structure was rebuilt. The
CQSUNLCK data object was nonrecoverable and a rebuild occurred after the data object was locked. The data object is now lost.

qpos One-byte input parameter that indicates the position on the queue to which the unlocked element is to be added.

X'00' Original client queue position. If the CQSREAD request that locked this data object read the first data object, this request unlocks the data object and adds it to beginning of the queue. If the CQSREAD request read the last data object, this request unlocks the data object and adds it to the end of the queue.

X'01' End of queue.

X'02' Beginning of queue.

locktoken Sixteen-byte input parameter that specifies the lock token that uniquely identifies the data object locked by a CQSREAD request. This parameter is required.

LISTVER=1 | listversion Input parameter that specifies an equate for the list version. Use the DSECT function of a CQSUNLCK request to include equate (EQU) statements in your program for the CQSUNLCK list versions.

PARM=parmaddress Four-byte input parameter that specifies the address of a parameter list used by the request to pass parameters to CQS. The length of the storage area must be at least equal to the EQU value CQSUNLCK_PARM_LEN (defined using the FUNC=DSECT request).

QTYPE=queuetype Four-byte input parameter that specifies the queue type from which the locked data objects were read. Valid values for the queue type are from 1 to 255 (decimal).

RETCODE=returncodeaddress Output parameter that specifies the address of a 4-byte field to contain the CQSUNLCK request return code.

If the return code in register 15 is nonzero, the values in the return and reason code fields are invalid, because the CQS interface detected an error and was unable to send the request to CQS.

RSNCODE=reasoncodeaddress Output parameter that specifies the address of a 4-byte field to contain the CQSUNLCK request reason code.

Return and Reason Codes for CQSUNLCK

| Table 47 lists the return and reason code combinations that can be returned for CQSUNLCK requests. Use a CQSUNLCK FUNC=DSECT request to include equate statements in your program for the return and reason codes. |

Table 47. CQSUNLCK Return and Reason Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000000'</td>
<td>X'00000000'</td>
<td>Request completed successfully.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000210'</td>
<td>cqstoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000214'</td>
<td>connecttoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000218'</td>
<td>FUNC is invalid.</td>
</tr>
</tbody>
</table>
Table 47. CQSUNLCK Return and Reason Codes (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000008'</td>
<td>X'00000240'</td>
<td>queuetype is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000244'</td>
<td>clientname is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000250'</td>
<td>count is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000254'</td>
<td>listaddress is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'0000027C'</td>
<td>CQSUNLCK is not allowed for a resource structure.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000284'</td>
<td>Parmlist version is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000288'</td>
<td>List version is invalid.</td>
</tr>
<tr>
<td>X'0000000C'</td>
<td>X'00000300'</td>
<td>Request succeeded for at least one but not all list entries. See compcode for individual errors.</td>
</tr>
<tr>
<td>X'0000000C'</td>
<td>X'00000304'</td>
<td>Request failed for all list entries. See compcode for individual errors.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000400'</td>
<td>A CQSYNC is required for this structure.</td>
</tr>
<tr>
<td>X'00000010'</td>
<td>X'00000430'</td>
<td>No CQS address space.</td>
</tr>
</tbody>
</table>

CQSUPD Request

Format for CQSUPD

DSECT Function of CQSUPD: Use the DSECT function of a CQSUPD request to include equate (EQU) statements in your program for the CQSUPD parameter list length, the CQSUPD return and reason codes, the CQSUPD parmlist version, and the CQSUPD list version.

```
CQSUPD—FUNC=DSECT
```

UPDATE Function of CQSUPD: Use the UPDATE function of a CQSUPD request to create or update one or more uniquely named resources on a resource structure. Each resource can optionally include a small client data area (DATA1) or a large client data area (DATA2).

```
CQSUPD—FUNC=UPDATE—CQSTOKEN=cqstokenaddress
```

```
CQSTOKEN—connecttokenaddress—PARM=parmaddress—LIST=resourcelistaddress
```

```
LISTSIZE=listsize—LISTVER=1—COUNT=resourcelistcount
```

```
ECB=ecbaddress—RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress
```

Usage of CQSUPD

A CQSUPD creates or updates one or more uniquely named resources on a resource structure. CQSUPD creates a resource if it does not exist, or updates a resource if it does exist. A resource can be created or updated with or without client data. Examples of resources include transactions and control blocks.
Parameter Description:

**CONTOKEN=connecttokenaddress**
Address of a 16-byte input parameter that specifies the connect token that uniquely identifies the client’s connection to a particular coupling facility structure managed by CQS. The connect token is returned by the CQSCONN request.

**COUNT=resourcelistcount**
Four-byte input parameter that specifies the number of entries in the list.

**CQSTOKEN=cqstokenaddress**
Address of a 16-byte input parameter that specifies the CQS registration token that uniquely identifies the client’s connection to CQS. The registration token is returned by the CQSREG request.

**ECB=ecbaddress**
Address of a 4-byte input parameter that specifies the MVS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise, it is processed synchronously.

**LISTSIZE=resourcelistsize**
Four-byte input parameter that specifies the size of the resource list. The list size must be specified because each entry in the list might have a variable length.

**LISTVER=1 | listversion**
Input parameter that specifies an equate for the list version. Use the DSECT function of a CQSUPD request to include equate (EQU) statements in your program for the CQSUPD list versions.

**LIST=resourcelistaddress**
Address of an input parameter that specifies a variable size resource list containing one or more entries. Each entry is a separate update request. Some fields in each entry must be initialized by the client prior to the CQSUPD request. Other fields are returned by CQS upon completion of the request.

The CQSUPDL list entry DSECT maps the list entries and can be used by the client. Multiple list entries must reside in contiguous storage.

Each list entry contains the following fields:

- **listentrylength**
  Four-byte input field that specifies the length of the list entry. The list entry length is variable, depending upon the data2 length, if specified. This parameter is required.

- **resourceid**
  Twelve-byte input field that contains the unique identifier of the resource to be created or updated on the resource structure. The resource identifier is unique in the IMSplex. The resource identifier consists of a 1-byte name type followed by an 11-byte client-defined resource name. The name type ensures uniqueness of client-defined names for resources with the same name type. Resources of different resource types can have the same name type. Valid values for the name type are decimal numbers from 1 to 255. The client-defined name has meaning to the client and consists of alphanumeric characters. This parameter is required.

- **resourcetype**
  One-byte field that specifies the resource type. The resource type is a client-defined physical grouping of resources on the resource structure.
Valid values for the resource type are decimal numbers from 1 to 255. If the resource type is greater than the maximum number of resource types defined by CQS (11), it is folded into one of the existing resource types. This parameter is required.

reserved
Three-byte reserved field.

options
Four-byte input field that specifies update options. This parameter is optional. Possible options are:

X'80000000'
Return data1 and owner, if update fails because of a version mismatch. This incurs the performance overhead of an additional CF access.

X'40000000'
Return data2, data1, and owner if update fails because of version mismatch. The data2 is returned if data2buffer and data2buffersize are specified. This incurs the performance overhead of an additional CF access.

X'20000000'
Delete data2.

compcode
Four-byte output field to receive the completion code from the request. Possible completion codes are:

X'00000000'
Request completed successfully.

X'00000004'
Request succeeded successfully, but only partial data returned in data2buffer.

X'00000020'
Resourceid is invalid. The name type must be a decimal number from 1 to 255.

X'00000024'
CQS internal error.

X'00000028'
Version doesn’t match that of existing resource.

X'00000030'
Resource already exists as a different name type.

X'00000034'
Structure is full.

X'00000038'
Resource type is invalid. The resource type must be a decimal number from 1 to 255.

X'0000003C'
Listentrylength is invalid. The list entry length must be a non-zero number greater than or equal to the minimum list entry length. See the CQSUPDL DSECT.

X'00000040'
Structure is inaccessible.
No CQS address space.

version
Eight-byte input and output field that specifies the version of a resource. The version is the number of times the resource has been updated. For the initial CQSUPD request to create the resource, version must be zero on input. For a subsequent CQSUPD request to update an existing resource, version must match the existing resource's version. The CQSUPD request increments the version by 1, updates the resource with the new version, and returns the new version as output. If a CQSUPD request to update an existing resource fails because of a version mismatch, CQS returns the correct version to the client as output. This parameter is required. If the data object is created, version is ignored on input and a version of 1 is returned as output.

owner
Eight-byte input and output field that specifies the owner of a resource. On input, owner is set for the resource. Specify zeroes to set no owner of a resource. Only one owner is permitted. If the update request fails because of a version mismatch and the option to return the owner is specified, the owner of the existing resource is returned as output. This parameter is required.

data1
Twenty-four byte input and output field that specifies data1, a small piece of client data for the resource to be updated. Specify zeroes to set no client data in data1. If the CQSUPD request fails because of a version mismatch and the option to return data1 is specified, data1 of the existing resource is returned as output. The performance of accessing the client data specified by data1 is faster than accessing client data specified by data2. This parameter is required.

data2size
Four-byte input and output field that specifies the size of client data data2 in data2buffer for the resource to be updated. Specify zero on input, if there is no data2 to update. If the CQSUPD request fails because of a version mismatch and the option to return data2 is set, the data2 size of the existing resource is returned as output. This parameter is optional.

data2buffersize
Four-byte input field that specifies the size of the data2buffer containing the client data data2 for the resource to be updated or returned as output. The maximum size that can be specified is 61312 bytes (X'EF80'). Specify zero if data2 does not need to be updated or returned as output. This parameter is optional.

data2buffer
Variable size input and output buffer that specifies data2, a large piece of client data for the resource to be updated. If the CQSUPD request fails because of a version mismatch and the option to return data2 is specified, data2 of the existing resource is returned, as much as fits into the data2buffer. This parameter is optional.

PARM=paraddress
Address of an input parameter list used by the request to pass parameters to CQS. The length of the storage area must be at least equal to the EQU value CQSUPD_PARM_LEN (defined using the FUNC=DSECT request).

RETCODE=returncodeaddress
Address of a four-byte output field to contain the CQSUPD return code. If the
Return code in register 15 is non-zero, the values returned for
returncodeaddress and reasoncodeaddress are not valid because CQS
detected an error and did not process the request.

RSNCODE=reasoncodeaddress
Address of a four-byte output field to contain the CQSUPD reason code.

Return and Reason Codes for CQSUPD

Table 48 table lists the return and reason codes that can be returned for CQSUPD
requests. Use a CQSUPD=DSECT request to include equate statements in your
program for the return and reason codes.

Table 48. CQSUPD Return and Reason Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00000000'</td>
<td>X'00000000'</td>
<td>Request completed successfully.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000210'</td>
<td>cqstoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000214'</td>
<td>contoken is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000218'</td>
<td>FUNC is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000250'</td>
<td>resourcelistcount is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000254'</td>
<td>listaddress is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000280'</td>
<td>Request not allowed for a queue structure.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000284'</td>
<td>Parmlist version is invalid.</td>
</tr>
<tr>
<td>X'00000008'</td>
<td>X'00000288'</td>
<td>List version is invalid.</td>
</tr>
</tbody>
</table>
| X'0000000C' | X'00000300' | Request succeeded for at least one but not all list
entries. See compcode for individual errors. |
| X'0000000C' | X'00000304' | Request failed for all entries. See compcode for
individual errors. |
| X'0000000C' | X'00000308' | Request failed for one or more list entries because of
version mismatch. Those resources already exist as
the resourcetype specified. All other entries were
successful. |
| X'00000010' | X'00000430' | No CQS address space. |
| X'00000014' | X'00000500' | Internal error. |

Example of Using a CQS Request: CQSREAD

Figure 25 on page 166 shows how you can use a CQSREAD request for a client
subsystem.
FUNCTION: USE CQSREAD REQUEST TO RETRIEVE A MESSAGE FROM SHARED QUEUES.

THE CALLER OF THIS MODULE PASSES THE ADDRESS AND SIZE OF A BUFFER. IF THIS MODULE ENDS WITH RC=0, THAT BUFFER HOLDS THE DATA OBJECT OR PARTIAL DATA. IF THIS MODULE ENDS WITH A NON-ZERO RC, THE BUFFER'S CONTENTS ARE UNPREDICTABLE.

REGISTERS ON ENTRY:

R2 - READ OBJECT BUFFER ADDRESS (BUFFER TO READ OBJECT INTO)
R3 - SIZE OF READ OBJECT BUFFER
R4 - CQS REGISTRATION TOKEN ADDRESS
R5 - CQS CONNECT TOKEN ADDRESS
R9 - ECB ADDRESS
R13 - SAVE AREA ADDRESS
R14 - RETURN ADDRESS
R15 - GEDOBJ ENTRY POINT ADDRESS

REGISTERS DURING EXECUTION:

R0 - WORK REGISTER
R1 - WORK REGISTER
R2 - CQSREAD PARMLIST AREA ADDRESS
R3 - WORK REGISTER
R4 - WORK REGISTER
R5 - WORK REGISTER
R6 - WORK REGISTER
R7 - WORK REGISTER
R8 - WORK REGISTER
R9 - ECB ADDRESS
R10 - WORK REGISTER
R11 - WORK REGISTER
R12 - BASE REGISTER
R13 - SAVE AREA ADDRESS
R14 - WORK REGISTER
R15 - WORK REGISTER

MACROS REFERENCED:

WAIT
CQSREAD

RETURN CODES:

R15 - RETURN CODE
X'00' CQSREAD SUCCESSFUL/PARTIAL DATA RETURNED
X'08' INTERFACE PROBLEM
X'0C' NO MESSAGE FOR QNAME
X'10' REQUEST IS UNSUCCESSFUL, UNEXPECTED RETURN OR REASON CODE
GETDOBJ CSECT
STM R14,R12,12(R13) SAVE THE REGS
LR R12,R15 R12 = PROGRAM BASE REGISTER
USING GETDOBJ,R12
LA R14,SAVEAREA CHAIN SAVE AREAS
ST R13,4(R14) THIS SAVEAREA BACKWARD PTR
ST R14,8(R13) LAST SAVEAREA FORWARD PTR
LA R13,SAVEAREA THIS ROUTINE'S SAVEAREA
ST R2,RDRBUFA SAVE A(BUFFER TO READ INTO)
ST R3,RDRBUFSZ SAVE READ BUFFER SIZE
MVC RDRROOTK,0(R4) SAVE COS REGISTRATION TOKEN
MVC RDRCONTK,0(R5) SAVE COS CONNECT TOKEN
ST R9,RDRECBA SAVE A(ECB)
LA R2,RDRPARM LOAD A(PARAMETER AREA) INTO R2
XC RDLCKTK,RDLCKTK LOCKTOKEN=0 FOR FIRST CQSREAD
XC 0(4,R9),0(R9) CLEAR CALLER'S ECB

****
* RETRIEVE RECORD FROM IMS SHARED QUEUES
****
CQSREAD FUNC=READ, X
  CQSTOKEN=(RDRROOTK), A(REGISTRATION TOKEN) X
  PARM=(R2), A(CQSREAD PARMLIST AREA) X
  CONTOKEN=(RDRCONTK), A(CONNECT TOKEN) X
  ECB=RDRECBA, A(ECB) X
  LCKTOKEN=(RDLCKTK), A(LOCK TOKEN) - RETURNED X
  UOW=(RDRUOW), A(UOW) - RETURNED X
  LOCAL=NO, READ OBJECT FROM SHARED QUEUE X
  QNAME=(RDRQNAME), A(QUEUE NAME) X
  QPOS=FIRST, READ FIRST OBJECT ON QUEUE X
  OBJSIZE=(RDROBJSZ), A(DATA OBJECT SIZE) - RETURNED X
  RSNCODE=(RDRRSN), A(REASON CODE) - RETURNED X
  RETCODE=(RDRRC), A(RETURN CODE) - RETURNED X
  BUFFER=RDRBUFA, A(CLIENT'S READ BUFFER) X
  BUFSIZE=(RDRBUFSZ) CLIENT'S READ BUFFER SIZE

LTR R15,R15 TEST RETURN CODE FROM CQS INTERFACE
BZ CHECKRC ZERO - CQSREAD OK
*
LA R15,R08 COS INTERFACE PROBLEM
B GOEXIT RETURN TO CALLER

****
* CHECK CQSREAD RETURN CODE
****
CHECKRC DS 0H WAIT ECB=(R9) WAIT FOR CQSREAD TO COMPLETE
L R15,RDRRC RETURN CODE
LTR R15,R15 CQSREAD REQUEST SUCCESSFUL?
BZ GOEXIT YES - RETURN TO CALLER

Figure 25. Sample for CQSREAD Request (Part 2 of 4)
****
* CHECK FOR CQS WARNING RETURN CODE
****
CLC RDRRC,=AL4(RQRCWARN) CQSREAD WARNING?
BNE UNEXPECT NO - SET RC AND RETURN TO CALLER
****
* CQSREAD: WARNING RETURN CODE - CHECK WARNING REASON CODE
* CHECK FOR DATA OBJECT
****
CLC RDRRSN,=AL4(RRDNOOBJ) NO DATA OBJECT?
BNE PARTIAL NO, CHECK NEXT REASON CODE
LA R15,RC0C SET NO DATA OBJECT RETURN CODE
B GOEXIT RETURN TO CALLER
****
* CHECK PARTIAL DATA RETURNED
* PARTIAL DATA RETURNED - RETURN DATA OBJECT - RETURN CODE 0
****
PARTIAL DS 0H
CLC RDRRSN,=AL4(RRDPARTL) PARTIAL DATA RETURNED?
BNE UNEXPECT NO - SET RC AND RETURN TO CALLER
LA R15,RC00 SET RETURN CODE
B GOEXIT RETURN TO CALLER
****
* UNEXPECTED RETURN OR REASON CODE
****
UNEXPECT DS 0H
LA R15,RC10 UNEXPECTED RETURN OR REASON CODE
B GOEXIT RETURN TO CALLER

***********************************************************************
* STANDARD EXIT *
***********************************************************************
GOEXIT DS 0H
L 13,4(,13) GET PREVIOUS SAVE LEVEL
L 14,12(13) A(RETURN-TO-CALLER)
LM 0,12,20(13) RESTORE REGS
OI 15(13),X'01' SET RETURN FLAG IN CALLER SAVE AREA
BR 14 RETURN TO CALLER

***********************************************************************
* CONSTANTS *
***********************************************************************

* GETDOBJ RETURN CODES
*
RC00 EQU 0 CQSREAD SUCCESSFUL -
RC08 EQU 8 INTERFACE PROBLEM
RC0C EQU 12 NO MESSAGE FOR QNAME
RC10 EQU 16 UNEXPECTED RETURN CODE

Figure 25. Sample for CQSREAD Request (Part 3 of 4)
* REGISTER EQUATES
*
R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15
***********************************************************************
*
VARIABLES
***********************************************************************
DS 0F
SAVEAREA DS 18F
DS 0D
RDRROTK DS XL16 CQS REGISTRATION TOKEN
RDRCONTK DS XL16 CQS CONNECT TOKEN
RDLCKTK DS XL16 LOCKTOKEN (RETURNED)
RDRUOW DS XL32 UOW (RETURNED)
RDRQNAME DS 0XL16 QUEUE NAME
DC X’05’ CLIENT QUEUE TYPE 5
DC CL15’FFSTRO1CF02CQ04’
RDROBSZ DS F OBJECT SIZE (RETURNED)
RDRSRN DS F CQSREAD REASON CODE (RETURNED)
RDRRC DS F CQSREAD RETURN CODE (RETURNED)
RDRBUFA DS A A(READ OBJECT BUFFER)
RDRBUFSZ DS F SIZE OF READ OBJECT BUFFER
RDRECBA DS A A(ECB)
RDRPARM DS XL(CQSREAD_PARM_LEN) CQSREAD PARMLIST
***********************************************************************
*
LITERALS
***********************************************************************
LTORG
CQSREAD FUNC=DSECT CQSREAD DSECTS & EQUATES
END GETDOBJ

Figure 25. Sample for CQSREAD Request (Part 4 of 4)
Example
Chapter 7. CQS Client Exit Routines

This chapter describes the Common Queue Server (CQS) client exit routines.

In this chapter:

- "Client CQS Event Exit Routine"
- "CQS Client Structure Event Exit Routine" on page 173
- "CQS Client Structure Inform Exit Routine" on page 181

This chapter contains Product-sensitive Programming Interface information.

CQS client exit routines allow a CQS client to monitor the CQS environment. They are written and supplied by a client (such as IMS). Each client must write its own exit routines tailored to the needs of that client product, to be supplied as part of the product. No sample CQS client exit routines are provided. The exit routines are given control in the client's address space in one of these two ways:

- For authorized clients (those running in supervisor state, key 0-7), the exits receive control in service request block (SRB) mode.
- For non-authorized clients (those running in problem state or non-key 0-7), the exits receive control as an IRB under the client TCB that owns the cross memory resources for the address space (the TCB pointed to by ASCBXTCB).

Because each call to a client exit routine runs under its own SRB, the order in which the exits are driven is not guaranteed. It is possible for client exit routines to be driven out of order (different from the order from which CQS scheduled them). Your exit routines must be able to tolerate events that are received out of order. All client exit routine parameter lists contain an 8-byte time stamp in STCK format that is the time when CQS scheduled the SRB for the exit routine. This time stamp may be used to help determine the original order of events.

### Client CQS Event Exit Routine

The CQS Event exit routine is driven when an event occurs in CQS that is related to CQS itself and may require some action to be taken by the client.

The client loads the exit routine and passes the exit routine address on the CQSREG request. This exit routine is driven in the client address space, either as an SRB (for authorized clients), or as an IRB (for non-authorized clients). The CQS Event exit routine is required.

The following CQS events drive the CQS Event exit routine:

- CQS initialization - client can reconnect to CQS
- CQS termination - abnormal termination

### Contents of Registers on Entry

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Length in bytes of the parameter list pointed to by R1.</td>
</tr>
<tr>
<td>1</td>
<td>Address of CQS Event Exit Parameter List (mapped by macro CQSCEVX).</td>
</tr>
<tr>
<td>13</td>
<td>Address of a standard 18-word save area, immediately followed by</td>
</tr>
</tbody>
</table>
CQS Event Client Exit

an 18-word work area that is available for the exit routine’s use. The save area and the work area are not chained together. The save area or work area storage is not cleared on entry to the CQS Event exit routine.

14 Return address.
15 Entry point of exit routine.

Restriction: All addresses passed to the CQS Event Exit routine are valid only until the exit routine returns to its caller. These addresses should never be stored and used after the CQS Event exit routine has returned. Doing so can cause unpredictable results, because the storage pointed to by the addresses may have changed, or it may have been freed.

Contents of Registers on Exit

The CQS Event exit routine must preserve the contents of R13; it does not need to preserve any other register’s contents. Therefore, it is free to use the save area pointed to by R13 for any calls to other services as needed (it can also use the 18-word area following the save area for additional save area or work area storage).

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>The same value it had on entry to the CQS Event exit routine.</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
<tr>
<td>0</td>
<td>Always set this to zero.</td>
</tr>
</tbody>
</table>

CQS Restart Entry Parameter List

Table 49 describes the CQS restart entry parameters for the Client CQS Event exit routine.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEVX_PVSN</td>
<td>X’00’</td>
<td>X’04’</td>
<td>Parameter List Version Number (00000001).</td>
</tr>
<tr>
<td>CEVX_EVENT</td>
<td>X’04’</td>
<td>X’04’</td>
<td>CQS Event Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 CQS Initialization Event (CEVX_INIT).</td>
</tr>
<tr>
<td>CEVX_SCODE</td>
<td>X’08’</td>
<td>X’04’</td>
<td>CQS Event Subcode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 Client can re-register and reconnect to CQS (CEVX_RESTART).</td>
</tr>
<tr>
<td>CEVX_DATA</td>
<td>X’0C’</td>
<td>X’04’</td>
<td>Event exit routine client data that was passed to CQS on the CQSREG request.</td>
</tr>
<tr>
<td>CEVX_CQSID</td>
<td>X’10’</td>
<td>X’08’</td>
<td>CQS identifier.</td>
</tr>
<tr>
<td>CEVX_CQSVER</td>
<td>X’18’</td>
<td>X’04’</td>
<td>CQS version number.</td>
</tr>
<tr>
<td>CEVX_TSTMP</td>
<td>X’1C’</td>
<td>X’08’</td>
<td>Time stamp representing the time the exit routine was scheduled (in STCK format).</td>
</tr>
</tbody>
</table>

CQS Abnormal Termination Parameter List

Table 50 on page 173 describes the CQS abnormal termination parameters for the Client CQS Event exit routine.
Table 50. Client CQS Event Exit Routine Parameter List: CQS Abnormal Termination

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>X'04'</td>
<td>Parameter List Version Number (00000001).</td>
</tr>
<tr>
<td>X'04'</td>
<td>X'04'</td>
<td>CQS Event Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 CQS Termination Event.</td>
</tr>
<tr>
<td>X'08'</td>
<td>X'04'</td>
<td>CQS Event Subcode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 CQS abnormal termination entry. The CQS address space is terminating abnormally.</td>
</tr>
<tr>
<td>X'0C'</td>
<td>X'04'</td>
<td>Event exit routine client data that was passed to CQS on the CQSREG request.</td>
</tr>
<tr>
<td>X'10'</td>
<td>X'08'</td>
<td>CQS identifier.</td>
</tr>
<tr>
<td>X'18'</td>
<td>X'04'</td>
<td>CQS version number.</td>
</tr>
<tr>
<td>X'1C'</td>
<td>X'08'</td>
<td>Time stamp representing the time the exit routine was scheduled (in STCK format).</td>
</tr>
<tr>
<td>X'24'</td>
<td>X'04'</td>
<td>Abnormal Termination reason code. (CQS abend code)</td>
</tr>
</tbody>
</table>

Client Processing after CQS Abnormal Termination or Restart

If a client is registered with CQS and CQS terminates abnormally, the client’s CQS Event exit routine is called with a CQS abnormal termination event. The client can choose to wait for CQS to be restarted, at which time the client’s CQS Event exit routine is scheduled for a CQS restart event. When the CQS restart event is received, the client must perform the following steps before it can resume making CQS requests:

1. The client must reregister with CQS using the CQSREG macro. This step is necessary to reestablish the cross-memory connections between the client and CQS. Failure to reregister can result in an S0D6 abend when the next CQS request is issued.
2. The client must reconnect, using the CQSCONN macro, to any structures it was using prior to the CQS failure.
3. The client must resync indoubt UOWs with CQS, using the CQSRSYNC macro.
4. The client must register interest in queues, using the CQSINFRM request. If CQS terminated abnormally, it lost all previous client registration information.

CQS Client Structure Event Exit Routine

The Client Structure Event exit routine is driven when an event occurs concerning a CQS-managed structure that may require some action to be taken by the client.

The client loads the exit routine and passes the address of the exit routine on the CQSCONN request. This exit routine is driven in the client address space, either as an SRB (for authorized clients), or as an IRB (for non-authorized clients). This exit routine is required, and applies both to resource and queue structures.

The following structure events drive the Client Structure Event exit routine:

- Resync UOW Processing
  - When CQS Resync processing completes for an individual UOW, which had been deferred.
  - When CQS Resync processing occurs for the list of client UOWs that were not passed during the CQS Resync request.
Structure Event Client Exit

- **Important:** Resync UOW Processing only applies to queue structures.

- **Checkpoint Event**
  - When structure checkpoint begin, end, or failure occurs.
  
  - **Important:** The Checkpoint event only applies to queue structures.

- **Structure Rebuild Event**
  - When structure copy (rebuild) begin, end, or failure occurs.
  - When structure recovery (rebuild) begin, end, or failure occurs.
  - When structure recovery lost UOWs occurs.

- **Structure Overflow Event**
  - When one or more queues move to the overflow structure.
  - When one or more queues move from the overflow structure. This event also indicates when the structure is no longer in overflow mode.

  - **Important:** The Structure Overflow event only applies to queue structures.

- **Structure Status Change Event**
  - When the structure is available again after a loss.
  - When the structure fails. For resource structures only, failure means that CQS cannot allocate a new resource structure.
  - When CQS is able to repopulate (allocate) a new resource structure.
  - When CQS loses its connection to the structure.
  - When the log stream becomes available, making the structure available.

### Contents of Registers on Entry

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Length in bytes of the parameter list pointed to by R1.</td>
</tr>
<tr>
<td>1</td>
<td>Address of Client Structure Event exit routine parameter list (mapped by macro CQSSEVX).</td>
</tr>
<tr>
<td>13</td>
<td>Address of a standard 18-word save area, immediately followed by an 18-word work area that is available for use by the exit routine. The save area and the work area are not chained together. The save area or work area storage is not cleared on entry to the Client Structure Event exit routine.</td>
</tr>
<tr>
<td>14</td>
<td>Return address.</td>
</tr>
<tr>
<td>15</td>
<td>Entry point of exit routine.</td>
</tr>
</tbody>
</table>

**Restriction:** All addresses that are passed to the Client Structure Event exit routine are valid only until the exit routine returns to its caller. These addresses should never be stored and used after the CQS Client Structure Event exit routine has returned. Doing so can cause unpredictable results, because the storage pointed to by the addresses may have changed, or it may have been freed.

### Contents of Registers on Exit

The Client Structure Event exit routine must preserve the contents of R13; it does not need to preserve any other register contents. Therefore, it is free to use the save area pointed to by R13 for any calls to other services as needed. The exit routine can also use the 18-word area following the save area for additional save area or work area storage.

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
</table>
Deferred Resync Complete Parameter List for CQS Client Structure Event

Table 51 describes the deferred resync complete parameters for the Client Structure Event exit routine.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>X'04'</td>
<td>Parameter List Version Number (00000001).</td>
</tr>
<tr>
<td>X'04'</td>
<td>X'04'</td>
<td>Structure Event Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Resync UOW Event.</td>
</tr>
<tr>
<td>X'08'</td>
<td>X'04'</td>
<td>Structure Event Subcode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Deferred Resync complete.</td>
</tr>
<tr>
<td>X'0C'</td>
<td>X'04'</td>
<td>Structure Event exit routine client data that was passed to CQS on the CQSCONN request.</td>
</tr>
<tr>
<td>X'10'</td>
<td>X'08'</td>
<td>CQS identifier.</td>
</tr>
<tr>
<td>X'18'</td>
<td>X'04'</td>
<td>CQS version number.</td>
</tr>
<tr>
<td>X'1C'</td>
<td>X'10'</td>
<td>Structure Name.</td>
</tr>
<tr>
<td>X'2C'</td>
<td>X'08'</td>
<td>Time stamp representing the time the exit routine was scheduled (in STCK format).</td>
</tr>
<tr>
<td>X'34'</td>
<td>X'20'</td>
<td>Unit of work (UOW) identifier.</td>
</tr>
<tr>
<td>X'54'</td>
<td>X'10'</td>
<td>Queue Name.</td>
</tr>
<tr>
<td>X'64'</td>
<td>X'10'</td>
<td>Deferred Resync token. This is the Put token that is used for Put Forget processing.</td>
</tr>
<tr>
<td>X'74'</td>
<td>X'02'</td>
<td>CQS UOW State</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'0010' Put Insync</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Client status is Put Complete. CQS status is Put Complete. CQS knows about the UOW and all data objects for the UOW are out on the coupling facility. A PUT token is returned for the UOW. The client should use the PUT token to issue the CQSPUT FUNC=FORGET request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'00F2' Unknown</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Client status is Put Complete. CQS has no knowledge of the UOW. If the client believes the UOW is in Put Complete status, the client must determine whether to reissue the CQSPUT requests.</td>
</tr>
<tr>
<td>X'76'</td>
<td>X'02'</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

CQS Resync Parameter List

Table 52 on page 176 describes the CQS initiated resync parameters for the Client Structure Event exit routine.
Table 52. Client Structure Event Routine Exit Parameter List: CQS Initiated Resync

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>X'04'</td>
<td>Parameter List Version Number (00000001).</td>
</tr>
<tr>
<td>X'04'</td>
<td>X'04'</td>
<td>Structure Event Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Resync UOW Event.</td>
</tr>
<tr>
<td>X'08'</td>
<td>X'04'</td>
<td>Structure Event Subcode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 CQS Initiated Resync processing.</td>
</tr>
<tr>
<td>X'0C'</td>
<td>X'04'</td>
<td>Structure Event exit routine client data that was passed to CQS on the CQSCCONN request.</td>
</tr>
<tr>
<td>X'10'</td>
<td>X'08'</td>
<td>CQS identifier.</td>
</tr>
<tr>
<td>X'18'</td>
<td>X'04'</td>
<td>CQS version number.</td>
</tr>
<tr>
<td>X'1C'</td>
<td>X'10'</td>
<td>Structure Name.</td>
</tr>
<tr>
<td>X'2C'</td>
<td>X'08'</td>
<td>Time stamp representing the time the exit routine was scheduled (in STCK format).</td>
</tr>
<tr>
<td>X'34'</td>
<td>X'04'</td>
<td>Number of unit of work (UOW) list entries.</td>
</tr>
<tr>
<td>X'38'</td>
<td>X'04'</td>
<td>Length of each UOW list entry.</td>
</tr>
<tr>
<td>X'3C'</td>
<td>X'04'</td>
<td>Offset into parmlist of start of UOW list. The parmlist is one contiguous piece of storage, including the UOW list.</td>
</tr>
</tbody>
</table>

CQS Resync UOW Entry

Table 53 describes the CQS resync UOW entry parameters for the Client Structure Event exit routine.

Table 53. CQS Resync UOW Entry Parameters

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>X'20'</td>
<td>Unit of work (UOW) identifier.</td>
</tr>
<tr>
<td>X'20'</td>
<td>X'10'</td>
<td>Queue name.</td>
</tr>
<tr>
<td>X'30'</td>
<td>X'10'</td>
<td>Resync token.</td>
</tr>
</tbody>
</table>

- If the CQS UOW status is locked, this field contains a lock token. This lock token is to be used on subsequent requests, such as CQSREAD and CQSUNLCK to process the locked data object.
- If the CQS UOW status is COLD QUEUE, this field contains a cold queue token. This cold queue token is to be used along with the UOW on a CQSRECVR request to recover the data object on the cold queue.
Table 53. CQS Resync UOW Entry Parameters (continued)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'40'</td>
<td>X'02'</td>
<td>CQS UOW Status</td>
</tr>
<tr>
<td>X'00F1'</td>
<td></td>
<td>Locked. This data object is locked. A lock token is passed back to the client in the Resync token field. This token field is required on subsequent requests to process the locked data object.</td>
</tr>
<tr>
<td>X'00F3'</td>
<td></td>
<td>Cold Queue: CQS-Client Cold Start. This data object is on the cold queue because of either a CQS cold start or client cold start. A cold queue token is passed back to the client in the Resync token field. This token field is required on a subsequent CQSRECVR request to process the data object on the cold queue.</td>
</tr>
<tr>
<td>X'00F4'</td>
<td></td>
<td>Cold Queue: Unknown. This data object is on the cold queue. CQS warm started after a structure rebuild from the log took place and the object was found locked by CQS. A cold queue token is passed back to the client in the Resync token field. This token field is required on a subsequent CQSRECVR request to process the data object on the cold queue.</td>
</tr>
<tr>
<td>X'42'</td>
<td>X'02'</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

Table 54 describes the checkpoint parameters for the Client Structure Event exit routine.

Table 54. Client Structure Event Exit Routine Parameter List: Checkpoint

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>X'04'</td>
<td>Parameter List Version Number (00000001).</td>
</tr>
<tr>
<td>X'04'</td>
<td>X'04'</td>
<td>Structure Event Code</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Checkpoint Event.</td>
</tr>
<tr>
<td>X'08'</td>
<td>X'04'</td>
<td>Structure Event Subcode</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Structure checkpoint begin.</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Structure checkpoint end.</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Structure checkpoint failure.</td>
</tr>
<tr>
<td>X'0C'</td>
<td>X'04'</td>
<td>Structure Event exit routine client data that was passed to CQS on the CQSCONN request.</td>
</tr>
<tr>
<td>X'10'</td>
<td>X'08'</td>
<td>CQS identifier.</td>
</tr>
<tr>
<td>X'18'</td>
<td>X'04'</td>
<td>CQS version number.</td>
</tr>
<tr>
<td>X'1C'</td>
<td>X'10'</td>
<td>Structure Name.</td>
</tr>
<tr>
<td>X'2C'</td>
<td>X'08'</td>
<td>Time stamp representing the time the exit routine was scheduled (in STCK format).</td>
</tr>
<tr>
<td>X'34'</td>
<td>X'08'</td>
<td>CQS identifier of the master CQS performing the checkpoint process.</td>
</tr>
</tbody>
</table>
### Structure Event Client Exit

**Table 54. Client Structure Event Exit Routine Parameter List: Checkpoint (continued)**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'3C'</td>
<td>X'01'</td>
<td>Flag byte.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'80'</strong></td>
</tr>
<tr>
<td>X'3D'</td>
<td>X'03'</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

### Structure Rebuild Parameter List for CQS Client Structure Event

**Table 55 describes the structure rebuild parameters for the Client Structure Event exit routine.**

**Table 55. Client Structure Event Exit Routine Parameter List: Structure Rebuild**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>X'04'</td>
<td>Parameter List Version Number (00000001).</td>
</tr>
<tr>
<td>X'04'</td>
<td>X'04'</td>
<td>Structure Event Code</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>3</strong> Structure Rebuild Event.</td>
</tr>
<tr>
<td>X'08'</td>
<td>X'04'</td>
<td>Structure Event Subcode</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>1</strong> Structure rebuild begin.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>2</strong> Structure rebuild (copy) end.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>3</strong> Structure rebuild (copy) failure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>4</strong> Structure rebuild failure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>5</strong> Structure rebuild (recovery) end.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>6</strong> Structure rebuild (recovery) failure.</td>
</tr>
<tr>
<td>X'0C'</td>
<td>X'04'</td>
<td>Structure Event exit routine client data that was passed to CQS on the CQSCONN request.</td>
</tr>
<tr>
<td>X'10'</td>
<td>X'08'</td>
<td>CQS identifier.</td>
</tr>
<tr>
<td>X'18'</td>
<td>X'04'</td>
<td>CQS version number.</td>
</tr>
<tr>
<td>X'1C'</td>
<td>X'10'</td>
<td>Structure Name.</td>
</tr>
<tr>
<td>X'2C'</td>
<td>X'08'</td>
<td>Time stamp representing the time the exit routine was scheduled (in STCK format).</td>
</tr>
<tr>
<td>X'34'</td>
<td>X'08'</td>
<td>CQS identifier of the master CQS performing the rebuild process.</td>
</tr>
<tr>
<td>X'3C'</td>
<td>X'01'</td>
<td>Flag byte.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>X'80'</strong></td>
</tr>
<tr>
<td>X'3D'</td>
<td>X'03'</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

### Structure Rebuild Lost UOWs Parameter List for CQS Client Structure Event

**Table 56 on page 179 describes the structure rebuild lost UOW parameters for the Client Structure Event exit routine. These UOWs are nonrecoverable and were lost by the last structure recovery. Some of the UOWs in the list may belong to other clients if the structure recovery occurred while CQS was down.**
Table 56. Client Structure Event Exit Routine Parameter List: Structure Rebuild Lost UOWs

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>X'04'</td>
<td>Parameter List Version Number (00000001).</td>
</tr>
<tr>
<td>X'04'</td>
<td>X'04'</td>
<td>Structure Event Code</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Structure Rebuild Event.</td>
</tr>
<tr>
<td>X'08'</td>
<td>X'04'</td>
<td>Structure Event Subcode</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Structure recovery lost UOWs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Important: This subcode applies only to queue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>structures.</td>
</tr>
<tr>
<td>X'0C'</td>
<td>X'04'</td>
<td>Structure Event exit routine client data that</td>
</tr>
<tr>
<td></td>
<td></td>
<td>was passed to CQS on the CQSCONN request.</td>
</tr>
<tr>
<td>X'10'</td>
<td>X'08'</td>
<td>CQS identifier.</td>
</tr>
<tr>
<td>X'18'</td>
<td>X'04'</td>
<td>CQS version number.</td>
</tr>
<tr>
<td>X'1C'</td>
<td>X'10'</td>
<td>Structure Name.</td>
</tr>
<tr>
<td>X'2C'</td>
<td>X'08'</td>
<td>Time stamp representing the time the exit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>routine was scheduled (in STCK format).</td>
</tr>
<tr>
<td>X'34'</td>
<td>X'08'</td>
<td>CQS identifier of the master CQS performing the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rebuild process.</td>
</tr>
<tr>
<td>X'3C'</td>
<td>X'01'</td>
<td>Flag byte.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'80'   Lost UOW was on client queue.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'40'   Lost UOW was locked.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'20'   Lost UOW was on COLDQ.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'10'   Lost UOW was on CQS private queue.</td>
</tr>
<tr>
<td>X'3D'</td>
<td>X'03'</td>
<td>Reserved.</td>
</tr>
<tr>
<td>X'40'</td>
<td>X'04'</td>
<td>Number of Lost UOW list entries.</td>
</tr>
<tr>
<td>X'44'</td>
<td>X'04'</td>
<td>Length of each Lost UOW list entry.</td>
</tr>
<tr>
<td>X'48'</td>
<td>X'04'</td>
<td>Offset into parmlist of start of Lost UOW list.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The parmlist is one contiguous piece of storage,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>including the Lost UOW list.</td>
</tr>
</tbody>
</table>

Rebuild Lost UOW Entry for CQS Client Structure Event

Table 57 describes the CQS rebuild lost UOW entry parameters for the Client Structure Event exit routine.

Table 57. CQS Rebuild Lost UOW Entry Parameters

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>X'20'</td>
<td>Unit of work (UOW) identifier.</td>
</tr>
<tr>
<td>X'20'</td>
<td>X'10'</td>
<td>Client Queue Name.</td>
</tr>
<tr>
<td>X'30'</td>
<td>X'1'</td>
<td>Lost UOW status.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'80'   Lost UOW was on client queue.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'40'   Lost UOW was locked.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'20'   Lost UOW was on COLDQ.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X'10'   Lost UOW was on CQS private queue.</td>
</tr>
<tr>
<td>X'31'</td>
<td>X'3'</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>
Structure Event Client Exit

Structure Overflow Parameter List for CQS Client Structure Event

Table 58 describes the structure overflow parameters for the Client Structure Event exit routine.

Table 58. Client Structure Event Exit Routine Parameter List: Structure Overflow

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>X'04'</td>
<td>Parameter List Version Number (00000001).</td>
</tr>
<tr>
<td>X'04'</td>
<td>X'04'</td>
<td>Structure Event Code</td>
</tr>
<tr>
<td>X'08'</td>
<td>X'04'</td>
<td>Structure Event Subcode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Move queues to overflow. One or more queues was selected as candidates to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>be moved to the overflow structure and was approved by the Queue Overflow</td>
</tr>
<tr>
<td></td>
<td></td>
<td>user exit routine.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 Move queues from overflow. One or more queues moved from the overflow</td>
</tr>
<tr>
<td></td>
<td></td>
<td>structure back to the primary structure, because the queues were drained on</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the overflow structure. New work for these queues is placed on the primary</td>
</tr>
<tr>
<td>X'0C'</td>
<td>X'04'</td>
<td>Structure Event exit routine client data that was passed to CQS on the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CQSCONN request.</td>
</tr>
<tr>
<td>X'10'</td>
<td>X'08'</td>
<td>CQS identifier.</td>
</tr>
<tr>
<td>X'18'</td>
<td>X'04'</td>
<td>CQS version number.</td>
</tr>
<tr>
<td>X'1C'</td>
<td>X'10'</td>
<td>Structure Name.</td>
</tr>
<tr>
<td>X'2C'</td>
<td>X'08'</td>
<td>Time stamp representing the time the exit routine was scheduled (in STCK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>format).</td>
</tr>
<tr>
<td>X'34'</td>
<td>X'08'</td>
<td>CQS identifier of the master CQS performing the overflow process.</td>
</tr>
<tr>
<td>X'3C'</td>
<td>X'01'</td>
<td>Flag byte.</td>
</tr>
<tr>
<td></td>
<td>X'80'</td>
<td>This CQS is the master of the process. The CQS identifier and master CQS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>identifier are the same.</td>
</tr>
<tr>
<td></td>
<td>X'40'</td>
<td>The structure is no longer in overflow mode. This value applies only to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>subcode 2.</td>
</tr>
<tr>
<td>X'3D'</td>
<td>X'03'</td>
<td>Reserved.</td>
</tr>
<tr>
<td>X'40'</td>
<td>X'04'</td>
<td>Number of Queue Name entries in the list.</td>
</tr>
<tr>
<td>X'44'</td>
<td>X'04'</td>
<td>Length of each Queue Name list entry.</td>
</tr>
<tr>
<td>X'48'</td>
<td>X'04'</td>
<td>Offset into parmlist of start of Queue Name list. Each Queue Name list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>entry contains the 16-byte queue name of a queue that is being moved to the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>overflow structure. The parmlist is one contiguous piece of storage,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>including the Queue Name list.</td>
</tr>
</tbody>
</table>

Structure Status Change Parameter List for CQS Client Structure Event

Table 59 describes the structure status change parameters for the Client Structure Event exit routine.

Table 59. Client Structure Event Exit Routine Parameter List: Structure Status Change

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>X'04'</td>
<td>Parameter List Version Number (00000002).</td>
</tr>
</tbody>
</table>
### Structure Event Client Exit

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'04'</td>
<td>X'04'</td>
<td>Structure Event Code.</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Structure Status Change Event.</td>
</tr>
<tr>
<td>X'08'</td>
<td>X'04'</td>
<td>Structure Event Subcode</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Structure available again after a loss.</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>The structure failed.</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>CQS lost its connection to the structure (STXLCONN).</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>The log stream is becoming available, making the structure available (STXAVLOG).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Important:</strong> This subcode applies only to queue structures.</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>The log stream is becoming unavailable, making the structure unavailable (STXFLOG).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Important:</strong> This subcode applies only to queue structures.</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Structure repopulation required due to structure failure.</td>
</tr>
<tr>
<td>X'0C'</td>
<td>X'04'</td>
<td>Structure exit routine client data that was passed to CQS on the CQSCONN request.</td>
</tr>
<tr>
<td>X'10'</td>
<td>X'08'</td>
<td>CQS identifier.</td>
</tr>
<tr>
<td>X'18'</td>
<td>X'04'</td>
<td>CQS version number.</td>
</tr>
<tr>
<td>X'1C'</td>
<td>X'10'</td>
<td>Structure Name.</td>
</tr>
<tr>
<td>X'2C'</td>
<td>X'08'</td>
<td>Time stamp representing the time the exit routine was scheduled (in STCK format).</td>
</tr>
<tr>
<td>X'34'</td>
<td>X'01'</td>
<td>Structure type</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Queue structure</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Resource structure</td>
</tr>
<tr>
<td>X'38'</td>
<td>X'18'</td>
<td>Not used.</td>
</tr>
<tr>
<td>X'50'</td>
<td>X'08'</td>
<td>Structure version of new structure that requires repopulation, because old structure failed.</td>
</tr>
</tbody>
</table>

### CQS Client Structure Inform Exit Routine

The Structure Inform exit routine is scheduled when work is placed on a queue for which the client has registered interest with a CQSINFRM request and when a CQSINFRM request is issued specifying that the exit routine be driven if there is work on the queue. The exit routine is also scheduled whenever a queue goes from an empty to non-empty state (when the first data object for a queue is written to the structure). If additional data objects are added to the queue, the inform exit routine, which has already been run once, is not notified again while there are still data objects on the queue.

The client loads the exit routine and passes the address of the exit routine on the CQSCONN request. This exit routine is driven in the client address space, either as an SRB (for authorized clients), or as an IRB (for non-authorized clients).
Structure Inform Client Exit

Restriction: This exit routine does not apply to resource structures.

Important: This exit routine is optional; however, if it is not supplied, the client is not notified when work is placed on the queues.

Contents of Registers on Entry

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Length in bytes of the parameter list pointed to by R1.</td>
</tr>
<tr>
<td>1</td>
<td>Address of CQS Structure Inform Exit Parameter List (mapped by macro CQSINFX).</td>
</tr>
<tr>
<td>13</td>
<td>Address of a standard 18-word save area, immediately followed by an 18-word work area available for use by the exit routine. The save area and the work area are not chained together. The save area or work area storage is not cleared on entry to the Structure Inform Exit routine.</td>
</tr>
<tr>
<td>14</td>
<td>Return address.</td>
</tr>
<tr>
<td>15</td>
<td>Entry point of exit routine.</td>
</tr>
</tbody>
</table>

Restriction: All addresses that are passed to the CQS Structure Inform exit routine are valid only until the exit routine returns to its caller. These addresses should never be stored and used after the CQS Structure Inform exit routine has returned. Doing so can cause unpredictable results, because the storage pointed to by the addresses may have changed, or it may have been freed.

Contents of Registers on Exit

The CQS Structure Inform exit routine must preserve the contents of R13 and it does not need to preserve any other register’s contents. Therefore, it is free to use the save area pointed to by R13 for any calls to other services as needed. It may also use the 18-word area following the save area for additional save area or work area storage.

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Same value as it had on entry to the CQS Structure Inform exit routine.</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
<tr>
<td>0</td>
<td>Always set this to zero.</td>
</tr>
</tbody>
</table>

Structure Inform Parameter List for CQS Client Structure Inform

Table 60 describes the parameters for the Client Structure Inform exit routine.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'00'</td>
<td>X'04'</td>
<td>Parameter List Version Number (00000001).</td>
</tr>
<tr>
<td>X'04'</td>
<td>X'04'</td>
<td>Structure Inform exit routine client data that was passed to CQS on the CQSCONN request.</td>
</tr>
<tr>
<td>X'08'</td>
<td>X'08'</td>
<td>CQS identifier.</td>
</tr>
<tr>
<td>X'10'</td>
<td>X'04'</td>
<td>CQS version number.</td>
</tr>
<tr>
<td>X'14'</td>
<td>X'10'</td>
<td>Structure Name.</td>
</tr>
<tr>
<td>Offset</td>
<td>Length</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>X'24'</td>
<td>X'08'</td>
<td>Time stamp representing the time the exit routine was scheduled (in STCK format).</td>
</tr>
<tr>
<td>X'2C'</td>
<td>X'04'</td>
<td>Number of Queue Names entries in the list.</td>
</tr>
<tr>
<td>X'30'</td>
<td>X'04'</td>
<td>Length of each Queue Name list entry.</td>
</tr>
<tr>
<td>X'34'</td>
<td>X'04'</td>
<td>Offset into parm list of start of Queue Name list. Each Queue Name entry in the list contains the 16-byte queue name for which a message has been queued. The parm list is one contiguous piece of storage, including the Queue Name list.</td>
</tr>
</tbody>
</table>
Structure Inform Client Exit
Chapter 8. CQS Diagnosis

This chapter describes diagnostic information that helps you analyze problems in CQS.

In this chapter:
- "CQS Log Records"
- "Printing CQS Log Records" on page 187

CQS Log Records

CQS writes records to the MVS log stream that contains all CQS log records from all CQSs that are connected to a structure pair. You can use the log records to:

- Diagnose problems related to the CQS address space.
- For CQS internal errors, The IBM support representative will direct you to print the appropriate log records.
- You can sometimes use information in the log records to set up a keyword string to search APAR descriptions and compare them to your own problem.
- Generate various reports related to the CQS address space, such as statistics about the number of requests.
- By knowing the content and format of the log records, you can set up a DFSERA10 job to format and print the specific log records you want.
- Restart CQS and recover shared queues, if necessary

Each CQS log record contains a log record prefix, followed by data that is unique to the record. Macro CQSLGRFX maps the log record prefix.

You can view the CQS log record formats by assembling mapping macro CQSLGREC with TYPE=ALL.

Table 61 shows the CQS log records. For each CQS log record, the table lists:
- The log record type and subtype
- The macro that maps the record
- The events that cause the record to be written

<table>
<thead>
<tr>
<th>Type</th>
<th>Subtype</th>
<th>Mapping Macro</th>
<th>Conditions for Writing the Log Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'03'</td>
<td>X'01'</td>
<td>CQSLGCON</td>
<td>CQSCCONN request: The client connect to a structure completed.</td>
</tr>
<tr>
<td>X'04'</td>
<td>X'01'</td>
<td>CQSLGDSC</td>
<td>CQSDISC request: The client disconnect from a structure completed.</td>
</tr>
<tr>
<td>X'07'</td>
<td>X'01'</td>
<td>CQSLGPUT</td>
<td>CQSPUT OBJECT request completed.</td>
</tr>
<tr>
<td></td>
<td>X'02'</td>
<td></td>
<td>CQSPUT COMMIT request completed.</td>
</tr>
<tr>
<td></td>
<td>X'03'</td>
<td></td>
<td>CQSPUT START request completed.</td>
</tr>
<tr>
<td></td>
<td>X'04'</td>
<td></td>
<td>CQSPUT FORGET request completed.</td>
</tr>
<tr>
<td></td>
<td>X'05'</td>
<td></td>
<td>CQSPUT ABORT request completed.</td>
</tr>
<tr>
<td></td>
<td>X'06'</td>
<td></td>
<td>CQSPUT request failed.</td>
</tr>
<tr>
<td></td>
<td>X'07'</td>
<td></td>
<td>CQSPUT system checkpoint record was written.</td>
</tr>
<tr>
<td></td>
<td>X'08'</td>
<td></td>
<td>CQSPUT FORGET request completed. This is a batched log record.</td>
</tr>
<tr>
<td>X'08'</td>
<td>X'01'</td>
<td>CQSLGRD</td>
<td>CSREAD request completed.</td>
</tr>
<tr>
<td></td>
<td>X'02'</td>
<td></td>
<td>CSREAD request failed.</td>
</tr>
<tr>
<td></td>
<td>X'03'</td>
<td></td>
<td>CSREAD system checkpoint record was written.</td>
</tr>
</tbody>
</table>
## CQS Log Records

<table>
<thead>
<tr>
<th>Type</th>
<th>Subtype</th>
<th>Mapping Macro</th>
<th>Conditions for Writing the Log Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'0B'</td>
<td>X'01'</td>
<td>CQSLGMOV</td>
<td>CQSMOVE or CQSUNLCK request completed.</td>
</tr>
<tr>
<td></td>
<td>X'02'</td>
<td></td>
<td>CQSMOVE or CQSUNLCK request failed.</td>
</tr>
<tr>
<td></td>
<td>X'03'</td>
<td></td>
<td>CQSMOVE or CQSUNLCK request moved an object between the primary and overflow structure.</td>
</tr>
<tr>
<td>X'0D'</td>
<td>X'01'</td>
<td>CQSLGDEL</td>
<td>CQSDEL request: Delete-type 1 (delete by token) completed.</td>
</tr>
<tr>
<td></td>
<td>X'02'</td>
<td></td>
<td>CQSDEL request: Delete-type 2 (delete by queue name) completed.</td>
</tr>
<tr>
<td></td>
<td>X'03'</td>
<td></td>
<td>CQSDEL request: Delete-type 3 (delete by queue name and UOW) completed.</td>
</tr>
<tr>
<td></td>
<td>X'04'</td>
<td></td>
<td>CQSDEL request: Delete-type 1 (delete by token) completed. This is a batched log record.</td>
</tr>
<tr>
<td>X'10'</td>
<td>X'01'</td>
<td>CQSLGSHT</td>
<td>CQSSHUT request completed.</td>
</tr>
<tr>
<td>X'32'</td>
<td>X'01'</td>
<td>CQSLGYCH</td>
<td>System checkpoint started.</td>
</tr>
<tr>
<td></td>
<td>X'02'</td>
<td></td>
<td>System checkpoint ended.</td>
</tr>
<tr>
<td></td>
<td>X'03'</td>
<td></td>
<td>System checkpoint failed.</td>
</tr>
<tr>
<td>X'40'</td>
<td>X'01'</td>
<td>CQSLGIST</td>
<td>Beginning of log stream.</td>
</tr>
<tr>
<td>X'42'</td>
<td>X'01'</td>
<td>CQSLGTCH</td>
<td>Structure checkpoint started.</td>
</tr>
<tr>
<td></td>
<td>X'02'</td>
<td></td>
<td>Structure checkpoint ended.</td>
</tr>
<tr>
<td></td>
<td>X'03'</td>
<td></td>
<td>Structure checkpoint failed.</td>
</tr>
<tr>
<td>X'43'</td>
<td>X'01'</td>
<td>CQSLGRBL</td>
<td>Structure rebuild started. Statistics about the old structure, the rebuild structure, and rebuild failure are mapped by CQSSSTT6.</td>
</tr>
<tr>
<td></td>
<td>X'02'</td>
<td></td>
<td>Structure rebuild ended. Statistics about the old structure, the rebuild structure, and rebuild failure are mapped by CQSSSTT6.</td>
</tr>
<tr>
<td></td>
<td>X'03'</td>
<td></td>
<td>Structure rebuild failed. Statistics about the old structure, the rebuild structure, and rebuild failure are mapped by CQSSSTT6.</td>
</tr>
<tr>
<td></td>
<td>X'04'</td>
<td></td>
<td>Structure rebuild resulted in a lost UOW list. This record lists the lost UOWs.</td>
</tr>
<tr>
<td>X'44'</td>
<td>X'01'</td>
<td>CQSLGOFI</td>
<td>Overflow threshold began.</td>
</tr>
<tr>
<td></td>
<td>X'02'</td>
<td></td>
<td>Overflow threshold ended.</td>
</tr>
<tr>
<td></td>
<td>X'03'</td>
<td></td>
<td>Overflow threshold failed.</td>
</tr>
<tr>
<td></td>
<td>X'04'</td>
<td></td>
<td>Overflow mode ended.</td>
</tr>
<tr>
<td></td>
<td>X'06'</td>
<td></td>
<td>Qnames were moved to overflow.</td>
</tr>
<tr>
<td></td>
<td>X'07'</td>
<td></td>
<td>Qnames were removed from overflow.</td>
</tr>
<tr>
<td></td>
<td>X'08'</td>
<td></td>
<td>CQSOVERFLOWQNMR, a control list entry containing the list of queue names deleted from overflow, was deleted.</td>
</tr>
<tr>
<td></td>
<td>X'09'</td>
<td></td>
<td>Overflow Scan Begin.</td>
</tr>
<tr>
<td></td>
<td>X'0A'</td>
<td></td>
<td>Overflow Scan End.</td>
</tr>
<tr>
<td></td>
<td>X'0B'</td>
<td></td>
<td>Private Queue Scan Begin.</td>
</tr>
<tr>
<td></td>
<td>X'0C'</td>
<td></td>
<td>Structure to be deleted.</td>
</tr>
</tbody>
</table>
Printing CQS Log Records

To print the CQS log records from the MVS system log, use the IMS File Select and Formatting Print utility (DFSERA10) with exit routine CQSERA30. The following example shows the required JCL to print the log records from an MVS system log. This JCL causes the MVS logger to invoke the default log stream subsystem exit routine, IXGSEXIT, to copy the log records. The exit routine returns a maximum of 32760 bytes of data for each log record even though CQS supports larger log records. You can specify the name of a different exit routine, if necessary.

**Example:** Use the JCL shown in [Figure 26](#) to print the CQS log records:

```plaintext
//CQSEERA10 JOB MSGLEVEL=1,MSGCLASS=A,CLASS=K
//STEP1 EXEC PGM=DFSERA10
//STEPLIB DD DISP=SHR,DSN=IMS.SDFSRESL
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=SYSLOG.MSGQ01.LOG,
//        SUBSYS=(LOGR,IXGSEXIT),
//        DCB=(BLKSIZE=32760)
//SYSIN DD *
CONTROL CNTL H=EOF
OPTION PRINT EXTR=CQSERA30
END

Figure 26. JCL to Print CQS Log Records
```

**DD Statements for CQS Diagnosis**

- **STEPLIB**
  
  DSN= points to IMS.SDFSRESL, which contains the IMS File Select and Formatting Print utility, DFSERA10.

- **SYSUT1**
  
  DSN= points to the CQS log stream name that was specified in the LOGNAME= parameter in the CQSSGxxx PROCLIB member.

**Control Statements for CQS Diagnosis**

- **H=**
  
  Specifies the number of log records to print. H=EOF prints all log records.

- **EXITR=CQSERA30**
  
  The CQS log record routine that is called to format each log record. This routine prints the record type and time-stamp information for each record, and dumps the contents of the record (up to a maximum of 32760 bytes (X'7FF8')).
Printing CQS Log Records

Related Reading: For a complete description of the IMS File Select and Formatting Print utility, see [IMS Version 8: Utilities Reference: System](#).

For a complete description of the MVS logger subsystem exit (IXGSEXIT) usage and parameters, see MVS Diagnostics: Tools and Service Aids, GA22-7589.

### Limiting Log Data to a Specified Time Range for CQS Diagnosis

You can limit the log records you print to those in a particular interval of time using the FROM and TO parameters on the SUBSYS statement. The DD card in Figure 27 illustrates this:

```
//SYSUT1 DD DSN=SYSLOG.MSGQ01.LOG,
//     SUBSYS=(LOGR,IXGSEXIT,
//          'FROM=(2001/042,11:00:00),TO=(2001/042,12:00:00)'),
//     DCB=(BLKSIZE=32760)
```

*Figure 27. DD Card to Limit Log Records that are Printed*

The DD card in Figure 27 would pass only log records from 11:00 to 12:00 on day 42 of the year 2001 to the DFSERA10 program. Dates and times specified in this manner are in GMT (Greenwich Mean Time). The seconds field of the time values is optional. If you want to use local dates and times, add the LOCAL keyword to the statement, as shown in Figure 28:

```
//SYSUT1 DD DSN=SYSLOG.MSGQ01.LOG,
//     SUBSYS=(LOGR,IXGSEXIT,
//          'FROM=(2001/042,11:00:00),TO=(2001/042,12:00:00),LOCAL'),
//     DCB=(BLKSIZE=32760)
```

*Figure 28. DD Card to Add Local Date and Time*

### Copying CQS Log Records for Diagnostics

IBM Software Support sometimes requires a copy of a range of CQS log records for problem determination. You can use the IEBGENER utility program to copy some or all of the CQS log for a structure to a BSAM data set to send to IBM Software Support. The copy made by IEBGENER is a binary image of the log records. The JCL in Figure 29 on page 189 copies CQS log records between 15:10 and 15:30 local time on day 89 of 2001 to a data set named CQS.LOG.COPY:
If you copy CQS log records using IEBGENER, be aware of the following:

- The copied records cannot be used by CQS in any way (such as restart or recovery). They are for diagnostic purposes only.
- CQS log records that are greater than 32 KB bytes in length will be truncated. The SUBSYS exit supports a maximum of a 32 KB record size.
Notices

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

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

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

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

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

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

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
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

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

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM’s application programming interfaces.
Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This publication is intended to help the customer perform the following tasks:

- Plan for and design the installation of Common Queue Server (CQS).
- Install and operate CQS.
- Diagnose and recover from CQS system problems.
- Write a CQS client.

The **IMS Version 8: Common Queue Server Guide and Reference** primarily documents Product-sensitive Programming Interface and Associated Guidance Information provided by IMS.

Product-sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of IMS. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

However, the **Common Queue Server Guide and Reference** also documents General-use Programming Interface and Associated Guidance Information and Diagnosis, Modification or Tuning Information provided by IMS.

General-use programming interfaces allow the customer to write programs that obtain the services of IMS.

General-use Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking: General-use Programming Interface and Associated Guidance Information ....

Diagnosis, Modification or Tuning Information is provided to help the customer diagnose, modify, or tune IMS.

**Attention:** Do not use this Diagnosis, Modification or Tuning Information as a programming interface.

Diagnosis, Modification or Tuning Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking: Diagnosis, Modification or Tuning Information ....
# Trademarks

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

<table>
<thead>
<tr>
<th>BookManager</th>
<th>NetView®</th>
</tr>
</thead>
<tbody>
<tr>
<td>CICS</td>
<td>OS/2®</td>
</tr>
<tr>
<td>DataPropagator™</td>
<td>OS/390</td>
</tr>
<tr>
<td>DB2</td>
<td>Parallel Sysplex</td>
</tr>
<tr>
<td>DB2 Universal Database™</td>
<td>Processor Resource/Systems Manager</td>
</tr>
<tr>
<td>IBM</td>
<td>PR/SM</td>
</tr>
<tr>
<td>IMS</td>
<td>RACF</td>
</tr>
<tr>
<td>IMS/ESA®</td>
<td>S/390</td>
</tr>
<tr>
<td>Language Environment</td>
<td>System/390</td>
</tr>
<tr>
<td>Library Reader</td>
<td>Tivoli®</td>
</tr>
<tr>
<td>MVS</td>
<td>WebSphere®</td>
</tr>
<tr>
<td>MVS/DFP</td>
<td>z/OS</td>
</tr>
<tr>
<td>MVS/ESA™</td>
<td></td>
</tr>
</tbody>
</table>

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc., in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

This bibliography lists all of the information in the IMS Version 8 library.

- **External Security Interface (RACROUTE) Macro Reference, GC28-1366**
- **DFSMs Access Method Services for Catalogs, SC26-7394**
- **MVS Initialization and Tuning Reference, SA22-7592**
- **MVS Programming: Authorized Assembler Services Guide, SA22-7608**
- **MVS Programming: Sysplex Services Guide, SA22-7617**
- **MVS System Commands, SA22-7627**
- **MVS Programming: Assembler Services Guide, SA22-7605**
- **MVS Setting Up a Sysplex, SA22-7625**
- **System/390 MVS: Sysplex Hardware and Software Migration, GC28-1210**

## IMS Version 8 Library

<table>
<thead>
<tr>
<th>Order Number</th>
<th>Acronym</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC27-1283</td>
<td>ADB</td>
<td>IMS Version 8: Administration Guide: Database Manager</td>
</tr>
<tr>
<td>SC27-1285</td>
<td>ATM</td>
<td>IMS Version 8: Administration Guide: Transaction Manager</td>
</tr>
<tr>
<td>SC27-1286</td>
<td>APDB</td>
<td>IMS Version 8: Application Programming: Database Manager</td>
</tr>
<tr>
<td>SC27-1288</td>
<td>APCICS</td>
<td>IMS Version 8: Application Programming: EXEC DLI Commands for CICS and IMS</td>
</tr>
<tr>
<td>SC27-1289</td>
<td>APTM</td>
<td>IMS Version 8: Application Programming: Transaction Manager</td>
</tr>
<tr>
<td>SC27-1290</td>
<td>BPE</td>
<td>IMS Version 8: Base Primitive Environment Guide and Reference</td>
</tr>
<tr>
<td>SC27-1291</td>
<td>CR</td>
<td>IMS Version 8: Command Reference</td>
</tr>
<tr>
<td>SC27-1292</td>
<td>CQS</td>
<td>IMS Version 8: Common Queue Server Guide and Reference</td>
</tr>
<tr>
<td>SC27-1293</td>
<td>CSL</td>
<td>IMS Version 8: Common Service Layer Guide and Reference</td>
</tr>
<tr>
<td>SC27-1294</td>
<td>CG</td>
<td>IMS Version 8: Customization Guide</td>
</tr>
<tr>
<td>SC27-1295</td>
<td>DBRC</td>
<td>IMS Version 8: Database Recovery Control (DBRC) Guide and Reference</td>
</tr>
<tr>
<td>LY37-3742</td>
<td>DGR</td>
<td>IMS Version 8: Diagnosis Guide and Reference</td>
</tr>
<tr>
<td>LY37-3743</td>
<td>FAST</td>
<td>IMS Version 8: Failure Analysis Structure Tables (FAST) for Dump Analysis</td>
</tr>
<tr>
<td>SC27-1296</td>
<td>JGR</td>
<td>IMS Version 8: IMS Java Guide and Reference</td>
</tr>
<tr>
<td>GC27-1297</td>
<td>IIV</td>
<td>IMS Version 8: Installation Volume 1: Installation Verification</td>
</tr>
<tr>
<td>GC27-1298</td>
<td>ISDT</td>
<td>IMS Version 8: Installation Volume 2: System Definition and Tailoring</td>
</tr>
<tr>
<td>SC27-1300</td>
<td>MIG</td>
<td>IMS Version 8: Master Index and Glossary</td>
</tr>
<tr>
<td>GC27-1301</td>
<td>MC1</td>
<td>IMS Version 8: Messages and Codes, Volume 1</td>
</tr>
<tr>
<td>GC27-1302</td>
<td>MC2</td>
<td>IMS Version 8: Messages and Codes, Volume 2</td>
</tr>
<tr>
<td>SC27-1303</td>
<td>OTMA</td>
<td>IMS Version 8: Open Transaction Manager Access Guide</td>
</tr>
<tr>
<td>SC27-1304</td>
<td>OG</td>
<td>IMS Version 8: Operations Guide</td>
</tr>
<tr>
<td>GC27-1305</td>
<td>RPG</td>
<td>IMS Version 8: Release Planning Guide</td>
</tr>
<tr>
<td>SC27-1308</td>
<td>URDBTM</td>
<td>IMS Version 8: Utilities Reference: Database and Transaction Manager</td>
</tr>
<tr>
<td>SC27-1309</td>
<td>URS</td>
<td>IMS Version 8: Utilities Reference: System</td>
</tr>
</tbody>
</table>

## Supplementary Publications

<table>
<thead>
<tr>
<th>Order Number</th>
<th>Acronym</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>GC27-1299</td>
<td>LPS</td>
<td>IMS Version 8: Licensed Program Specifications</td>
</tr>
<tr>
<td>SC27-1307</td>
<td>SOC</td>
<td>IMS Version 8: Summary of Operator Commands</td>
</tr>
</tbody>
</table>
### Publication Collections

<table>
<thead>
<tr>
<th>Order Number</th>
<th>Description</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>LK3T-7092</td>
<td>CD</td>
<td>IMS Version 8 Softcopy Library</td>
</tr>
<tr>
<td>LK3T-7144</td>
<td>CD</td>
<td>IMS Favorites</td>
</tr>
<tr>
<td>LBOF-7787</td>
<td>Hardcopy and CD</td>
<td>Licensed Bill of Forms (LBOF): IMS Version 8 Hardcopy and Softcopy Library</td>
</tr>
<tr>
<td>SBOF-7788</td>
<td>Hardcopy</td>
<td>Unlicensed Bill of Forms (SBOF): IMS Version 8 Unlicensed Hardcopy Library</td>
</tr>
<tr>
<td>SK2T-6700</td>
<td>CD</td>
<td>OS/390 Collection</td>
</tr>
<tr>
<td>SK3T-4270</td>
<td>CD</td>
<td>z/OS Software Products Collection</td>
</tr>
<tr>
<td>SK3T-4271</td>
<td>DVD</td>
<td>z/OS and Software Products DVD Collection</td>
</tr>
</tbody>
</table>

### Accessibility Titles Cited in this Book

<table>
<thead>
<tr>
<th>Order Number</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA22-7787</td>
<td>z/OS TSO Primer</td>
</tr>
<tr>
<td>SA22-7794</td>
<td>TSO/E User’s Guide</td>
</tr>
<tr>
<td>SC34-4822</td>
<td>ISPF User’s Guide, Volume 1</td>
</tr>
</tbody>
</table>
Index

A
abnormal termination or restart, client processing after 173
altering structures 38
assembling a client program 81
authorization
requests 74
authorizing CQS
connections 37
registration 37
Automatic Restart Manager
policy 36
using 35

B
Base Primitive Environment.
See BPE
benefits of using CQS 5
BPE (Base Primitive Environment) 1
defining 15
definition 1
BPE Statistics user exit 71

C
CFRM (coupling facility resource management)
couple data set format utility 47
policy 12
policy, defining 15, 38
CFSizer 15
checkpoint
structure, initiating 39
system, initiating 39
checkpoint data set 1, 39
checkpoints
client initiating 82
Client Connection user-supplied exit routine, CQS 53
client exit routines (CQS) 171
Event 171
Structure Event 173
Structure Inform 181
client program
assembling 81
writing 73
client requests 9
assembling a program 81
authorization 74
coding 74
CQSSBWSE 86
CQSCHKPT 93
CQSDEL 102
CQSDEREG 106
CQSDISC 107
CQSinFrm 112
CQSMOVE 116
CQS_PUT 120
client requests (continued)
CQSQUERY 127
CQSREAD 136
CQSRECVR 141
CQRSREG 146
CQSSSYNC 148
CQSSHUT 155
CQSUNLOCK 156
CQSUPD 161
ECB, using 78
environmental requirements 75
introduction 73
lists, using 79
literals, coding 78
parameters, coding 76
requests
CQSSCONN 96
return and reason codes 79
sequence of 74
Client Structure Event exit 173
parameters 175
Client Structure Inform exit 181
parameters 182
client, definition 1
coding requests 74
cold queue, definition 1
cold start 35
structures 33
Common Queue Server.
See CQS
Common Service Layer
definition 1
components of a CQS 5
copy
structures 46
copying log records 188
couple data set format utility 47
coupling facility, definition 1
CQS (Common Queue Server) requests.
See requests
CQS (Common Queue Server) 1, 11
administering 31
authorization 37
benefits 5
client connection
establishing 36
client exit routines
Event 171
Structure Event 173
Structure Inform 181
client failure 36
clients 165
components 5
data sets 27
defining 11, 15
definition 1
diagnosis 185

© Copyright IBM Corp. 1997, 2005
CQS (Common Queue Server) (continued)

- Channel common data set 27
- Execution data sets 27
- System checkpoint data set 27

Execution parameters:
- Description 17
- Specifying 17

Exit routines:
- See CQS user-supplied exit routines

Failure 36

Functions 6

Global structure definition PROCLIB member:
- Description 22
- Keywords 22
- Specifying 22

Information for restart 32

Initialization parameters PROCLIB member:
- Description 19
- Keywords 19
- Specifying 19

JCL for printing log records 187

Local structure definition PROCLIB member:
- Description 20
- Keywords 20
- Specifying 20

Log records 185

Multiple clients 14

Overview 4

Parameters:
- CQS 17
- CQS PROCLIB 19

Preparing to start 15

Printing log records 187

Rebuilding structures 43

Recovering 43

Restarting 36

- Cold start 35
- Description 34
- Warm start 34

Restarting information 32

Restarting structures 32

Allocation 32

Restating after system checkpoint 39

Shutting down 48

Starting 31

Structure cold start 33

Structure types managed 6

Structure warm start 32

Tailoring 11

CQS Event exit 171

- Abnormal termination 173
- Parameters 172
- Parameters, abnormal termination 172

CQS restart

- Definition 2

CQS statistics:
- Using BPE Statistics user exit 71

CQS structure overflow function 41

CQS terms 1

CQS user-supplied exit routines 51

CQS user-supplied exit routines (continued)

Client Connection
- General 53
- Parameters 54
- Register contents 54
- General information 51

Initialization-Termination (Init-Term)
- General 52
- Parameters 53
- Register contents 52

Queue Overflow
- General 55
- Parameters 56
- Register contents 56

Structure Event
- Checkpoint parameters 68
- Connection parameters 67
- General 66
- Overflow parameters 70
- Rebuild parameters 69
- Register contents 67
- Routine parameters 67
- Status change parameters 71

Structure Statistics
- CQS request statistics record 59
- Data object statistics record 60
- General 57
- MVS request statistics record 61
- Parameters 58
- Queue name statistics record 61
- Register contents 57
- Structure checkpoint statistics entry 65
- Structure checkpoint statistics record 64
- Structure process statistics record 59
- Structure rebuild statistics record 62

CQS-managed rebuild 44

CQSBRWSE request 86

- Functions 86
- Parameters 88
- Return and reason codes 92
- Syntax 86
- Usage 87

CQSCHKPT request

- Format 93
- Parameters 95
- Return and reason codes 96
- Syntax 93
- Usage 94

CQSCONN request

- Format 96
- Parameters 97
- Return and reason codes 101
- Syntax 96
- Usage 97

CQSCONND request

- Format 102
- Parameter 102
- Return and reason codes 105
- Syntax 102
- Usage 102
CQSDEREG request
  format 106
  parameters 106
  return and reason codes 107
  syntax 106
  usage 106
CQSDISC request
  format 107
  parameters 109
  return and reason codes 111
  syntax 107
  usage 108
CQSINFRM request
  format 112
  parameters 113
  return and reason codes 116
  syntax 112
  usage 113
CQSIxxx 19
CQSLOCAL, definition 2
CQSMOVE request
  format 116
  parameters 118
  return and reason codes 119
  syntax 116
  usage 117
CQSOUTPUT request
  actions 122
  format 120
  parameters 123
  return and reason codes 126
  syntax 120
  timestamp for shared queue message 122
  usage 121
CQSQUERY request
  format 127
  parameters 131
  return and reason codes 135
  syntax 127
  usage 129
CQSREAD request
  example 165
  format 136
  functions 136
  parameters 138
  return and reason codes 140
  syntax 136
  usage 137
CQSRECVCR request
  format 141
  functions 141
  parameters 143
  return and reason codes 145
  syntax 141
  usage 143
CQSREG request
  functions 146
  parameters 146
  return and reason codes 147
  syntax 146
  usage 146
CQSRSYNC request
  format 148
  functions 148
  parameters 150
  return and reason codes 153
  syntax 148
  usage 149
CQSSGxxx 22
CQSSHUT request
  format 155
  functions 155
  parameters 155
  return and reason codes 156
  syntax 155
  usage 155
CQSSLxxx 20
CQSUNLCK request
  format 156
  functions 156
  parameters 158
  return and reason codes 160
  syntax 156
  usage 157
CQSUPD request
  format 161
  functions 161
  parameters 162
  return and reason codes 165
  syntax 161
  usage 161
CSL (Common Service Layer)
  definition 1

D
data object, definition 2
data sets
  CQS execution 27
  structure recovery 28
  system checkpoint 27
defining BPE 15
defining CQS 15
defining MVS policies 11
defining policies 12
definitions of CQS terms 1
deleting structures 48
DFSEMA10 187

E
ECB (MVS event control block), using with client request 78
environments
  client requests 75
  events, handling 81
    cold start 81
  registering interest in queues 82
  shutting down CQS 82
  tuning for performance 82
exit routines
  client 171
exit routines (continued)

    Event 171
    user-supplied, CQS 51

F

functions of CQS 6

H

hardware requirements 9

I

IMSpex, definition 2

Initialization-Termination (Init-Term) user-supplied exit routine

    CQS 52

L

LEID (list entry ID), definition 2

limiting log data 188

list entry controls, definition 2

list entry ID (LEID), definition 2

list header number, definition 2

list header, definition 2

list structure, definition 2

lists, using with client request 79

literals 78

literals, using 78

local list entry 2

log records

    control statements for printing 187
    copying 188
    DD statements for printing 187
    description 185
    JCL for printing 187
    limiting log data 188
    printing 187
    table 185
    types 185
    viewing format 185

log token, definition 2

LOGR (system logger) policy 12

M

managing structure usage 41

master CQS, definition 2

message (continued)

    CQS0268I 25
    CQS0300I 49
    IXC585E 42
    IXC586I 42
    MVS Automatic Restart Manager 35
    MVS program properties table
        adding CQSINIT0 16
        updating 16

O

overflow structure, definition 2

P

parameter lists

    abnormal termination 172
    Client Connection user exit 54
    Client Disconnect user exit 54
    Initialization user exit 53
    Queue Overflow user exit 56, 57
    restart entry 172
    Structure Event exit routine
        checkpoint 177
        Deferred Resync Complete 175
        resync, CQS 175
        structure overflow 180
        structure rebuild 178
        structure rebuild lost UOWs 178
        structure status change 180
    Structure Event user exit 67
        checkpoint 68
        connect 67
        overflow 70
        Rebuild 69
        status change 71
    Structure Inform exit routine 182
    Structure Statistics user exit 58
    Termination user exit 53

performance tuning 82

planning for CQS

    hardware requirements 9
    software requirements 9

policies

    ARM 12
    CFRM 12
    defined 11
    defining 12
    LOGR 12
    SFM 12

preventing structure full 41

primary structure, definition 3

program, assembling 81

Q

queue name, definition 3

Queue Overflow user-supplied exit routine

    CQS 55

queue structures 6
queue type
  definition 3
queues
  definition 3
  object on the cold queue 82
  registering interest in 82
  types, definition 3

R
RACF 37
rebuild lost UOW entry, CQS 179
rebuilding structures 43
recovering CQS 43
recovery
  structures 45
  structures, for restart 33
recovery functions 8
register contents
  Client Connection user exit 54
  Client Structure Event exit 174
  Client Structure Inform exit 182
  CQS Event exit 171, 172
  Initialization-Termination user exit 52, 53
  Queue Overflow user exit 56
  Structure Event user exit 67
  Structure Statistics user exit 57, 58
register, using 77
registers
  client requests 76
  using 76
requests 73
  assembling a program 81
  authorization 74
  coding 74
  CQSBRWSE 86
  CQSCHKPT 93
  CQSCONN 96
  CQSDEL 102
  CQSDEREG 106
  CQSDISC 107
  CQSMINF 112
  CQSMOVE 116
  CQSPUT 120
  CQSQUERY 127
  CQSREAD 136
  CQSREG 141
  CQSERG 146
  CQSSSYNC 148
  CQSSHUT 155
  CQSSUNLCK 156
  CQSUPD 161
  DSECTs, using 81
  ECB, using 78
  environmental requirements 75
  example 165
  introduction 73
  lists, using 79
  literals, coding 78
  literals, using 78
  parameters, coding 76
requests (continued)
  register, using 77
  return and reason codes 79
  sample 165
  sequence of 74
  symbol name, using 77
  symbol value, using 77
requirements
  hardware 9
  software 9
resource structure, definition 3
resource structures 7
resource type
definition 3
restart
  MVS Automatic Restart Manager 35
  structure recovery 33
restarting CQS
  cold start 35
  description 34
  structure initialization 32
  warm start 34
resync UOW entry, CQS 176
return and reason codes
  client requests 79
  CQSBRWSE request 92
  CQSCHKPT request 96
  CQSCONN request 101
  CQSDEL request 105
  CQSDEREG request 107
  CQSDISC request 111
  CQSMINF request 116
  CQSMOVE request 119
  CQSPUT request 126
  CQSQUERY request 135
  CQSREAD request 140
  CQSRECVR request 145
  CQSREG request 147
  CQSSYNC request 153
  CQSSHUT request 156
  CQSSUNLCK request 160
  CQSUPD request 165
RM (Resource Manager), definition 3
routines
  client 171
  user-supplied, CQS 51
S
sequence of requests 74
setting up a sysplex 11
SFM (sysplex failure management) policy 12
shared queue
timestamp 122
shared queue, definition 3
shutting down CQS 48, 82
software requirements 9
special events, handling.
  See events, handling
starting CQS 15, 31
statistic records
  CQS request 61
data object 60
MVS request 61
queue name 61
request 59
structure checkpoint 64
structure checkpoint entry 65
structure process 59
structure rebuild 62
structure alter 38
structure checkpoint, initiating 39
structure copy 46
structure duplexing 8, 46
  enabling 47
Structure Event user-supplied exit routine
  CQS 66
structure full
  managing 41
  monitoring 42
    structure overflow 43
structure functions 7
structure overflow 7
  function 41
  structure full monitoring 43
structure pair
  definition 3
structure rebuild 8
  initiating 44
structure recovery 45
structure recovery data set, definition 4
structure repopulation 45
Structure Statistics user-supplied exit routine
  CQS 57
structure types 6
structure usage
  managing 41
structures
  authorizing connections to 37
  cold start 33
deleting 48
duplexing 46
empty 32
functions 7
initialization 32
rebuilding 44
recovery for restart 33
repopulation 45
resource 7
restarting CQS 32
size 15, 38
structure copy 46
structure recovery 45
types 6
  warm start 32
supporting multiple clients 14
symbol name, using 77
symbol value, using 77
sysplex, definition 4
system checkpoint, initiating 39
system-managed rebuild 43

t
  timestamp
    for shared queue message 122

U
  unit of work, definition 4
user exits (CQS).
    See CQS user-supplied exit routines
user-managed rebuild 44
user-supplied exit routines.
    See CQS user-supplied exit routines
UXPL_EXITPLP
  Client Connections exit 54
  Init-Term exit 53
  Queue Overflow exit 56
  Structure Statistics exit 58

W
  warm start 34
  warm starting structures 32
writing a CQS client 73