

IMS



Common Queue Server Guide and Reference

Version 9

IMS



Common Queue Server Guide and Reference

Version 9

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 185.

Quality Partnership Program (QPP) Edition (June 2004) (Softcopy Only)

This QPP edition replaces or makes obsolete the previous edition, ZES1-2339-01. 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, 2004. 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
Syntax Diagram Example	xiii
How to Send Your Comments	xiii
Summary of Changes	xv
Changes to the Current Edition of This Book for IMS Version 9	xv
Changes to This Book for IMS Version 9	xv
Library Changes for IMS Version 9.	xv
New and Revised Titles	xv
Terminology Changes	xv
Accessibility Enhancements	xvi
Chapter 1. Introduction	1
Common Queue Server Overview	1
CQS Benefits	2
CQS Components	2
CQS Functions	3
Structures Managed by CQS	3
CQS Structure Functions	4
CQS Recovery Functions	5
CQS Client Requests	6
Planning for CQS Hardware and Software Requirements	6
Chapter 2. CQS Definition and Tailoring	7
CQS As Part of a Sysplex	7
CQS and Defining z/OS Policies	7
CQS's Support of Multiple Clients	11
Determining Structure Size for CQS Connections	12
Preparing to Start CQS	12
Updating z/OS Program Properties Table for CQS	13
CQS Execution Parameters	14
CQS Initialization Parameters PROCLIB Member (CQSIPxxx)	16
CQS Local Structure Definition PROCLIB Member (CQSSLxxx)	17
CQS Global Structure Definition PROCLIB Member (CQSSGxxx)	19
CQS Execution Data Sets	24
CQS System Checkpoint Data Set.	24
CQS Structure Recovery Data Sets	25
Chapter 3. CQS Administration	27
Starting CQS	27
Recording Information Necessary for Starting CQS	28
Restarting CQS Structures	28
CQS Structure Allocation	28
CQS Structure Warm Start	28
CQS Structure Cold Start	29
CQS Structure Recovery for Restarting	29
Restarting CQS.	30

CQS Warm Start	30
CQS Cold Start.	31
Using the z/OS Automatic Restart Manager with CQS	31
Restarting CQS After CQS Resource Cleanup Failures	32
Establishing Client Connection to CQS During Failed Client Takeover.	32
Authorizing Access To CQS	33
Authorizing CQS Registration	33
Authorizing Connections to CQS Structures	33
Using Structure Alter for CQS	34
Using CQS System Checkpoint.	34
CQS Checkpoint Data Set.	35
How CQS Restarts after System Checkpoint	35
Using CQS Structure Checkpoint	35
Preventing CQS Structure Full	37
CQS Structure Overflow Function	37
CQS Structure Full Monitoring	38
Using Structure Full Monitoring with CQS Structure Overflow	39
Rebuilding Structures in CQS	39
z/OS System-Managed Rebuild and CQS	39
CQS-Managed Rebuild	40
Initiating Structure Rebuild with z/OS and CQS	40
CQS Structure Repopulation	41
CQS Structure Recovery	41
CQS Structure Copy	42
z/OS Structure Duplexing for CQS.	42
Deleting a Structure When CQS Is Not Connected.	44
Shutting Down CQS	44
Chapter 4. CQS User-Supplied Exit Routines	47
General User-Supplied Exit Routine Interface Information for CQS	47
CQS Initialization-Termination User-Supplied Exit Routine	48
Contents of Registers on Entry	48
Contents of Registers on Exit	48
CQS Initialization and Termination Parameter Lists.	49
CQS Client Connection User-Supplied Exit Routine	49
Contents of Registers on Entry	49
Contents of Registers on Exit	50
CQS Client Connection and Disconnect Parameter Lists	50
Queue Overflow User-Supplied Exit Routine for CQS.	51
Contents of Registers on Entry	52
Contents of Registers on Exit	52
CQS Queue Overflow Parameter List	52
CQS Structure Statistics User-Supplied Exit Routine	53
Contents of Registers on Entry	53
Contents of Registers on Exit	54
CQS Structure Statistics User-Supplied Exit Routine Parameter List	54
CQS Structure Process Statistics Record	55
CQS Request Statistics Record.	55
Data Object Statistics Record for CQS	56
Queue Name Statistics Record for CQS	57
z/OS Request Statistics Record for CQS	57
Structure Rebuild Statistics Record for CQS	58
Structure Checkpoint Statistics Record for CQS.	60
Structure Checkpoint Statistics Gathered by CQS	61
CQS Structure Event User-Supplied Exit Routine	62
Contents of Registers on Entry	63

Contents of Registers on Exit	63
Routine Parameter Lists	63
CQS Structure Event Exit Routine Parameter List	63
CQS Structure Event Exit Routine Checkpoint Parameter List.	64
CQS Structure Event Exit Routine Rebuild Parameter List	65
CQS Structure Event Exit Routine Overflow Parameter List	66
CQS Structure Event Exit Routine Status Change Parameter List	67
CQS Statistics Available through the BPE Statistics User Exit.	67
Chapter 5. Writing a CQS Client	69
Introducing CQS Client Requests	69
Sequence of CQS Requests Issued by a Client for Queue Structure	70
Coding CQS Requests	70
Authorization for CQS	70
Environmental Requirements for CQS	71
Using Registers with CQS Requests	72
Coding Parameters for CQS Requests	73
Using an ECB with CQS Requests	74
Using Lists in the CQS Requests	75
Return Codes and Reason Codes for CQS Requests.	75
Assembling a Program with CQS Requests	77
CQS Clients and Handling Special Events	77
CQS Cold Start.	77
Registering Interest in Queues with CQSINFRM.	78
Working with Objects on the Cold Queue using CQS Requests	78
Initiating Checkpoints using CQS Requests	78
Shutting Down CQS	78
Tuning to Improve CQS Performance	78
Chapter 6. CQS Client Requests	79
Using CQS Client Requests	79
CQSBrowse Request	80
CQSCONN Request	90
CQSDEL Request.	96
CQSDEREG Request	100
CQSDISC Request	101
CQSINFRM Request	106
CQSMOVE Request	110
CQSPUT Request	114
CQSQUERY Request	121
CQSREAD Request	130
CQSRECVR Request	135
CQSREG Request	140
CQSRSYNC Request	142
CQSSHUT Request	149
CQSUNLCK Request	150
CQSUPD Request	155
Example of Using a CQS Request: CQSREAD	159
Chapter 7. CQS Client Exit Routines.	165
Client CQS Event Exit Routine	165
Contents of Registers on Entry	165
Contents of Registers on Exit	166
CQS Restart Entry Parameter List	166
CQS Abnormal Termination Parameter List	166
Client Processing after CQS Abnormal Termination or Restart	167

CQS Client Structure Event Exit Routine	167
Contents of Registers on Entry	168
Contents of Registers on Exit	168
Deferred Resync Complete Parameter List for CQS Client Structure Event	169
CQS Resync Parameter List	169
CQS Resync UOW Entry.	170
Checkpoint Parameter List for CQS Client Structure Event	171
Structure Rebuild Parameter List for CQS Client Structure Event	172
Structure Rebuild Lost UOWs Parameter List for CQS Client Structure Event	172
Rebuild Lost UOW Entry for CQS Client Structure Event	173
Structure Overflow Parameter List for CQS Client Structure Event	174
Structure Status Change Parameter List for CQS Client Structure Event	174
CQS Client Structure Inform Exit Routine.	175
Contents of Registers on Entry	176
Contents of Registers on Exit	176
Structure Inform Parameter List for CQS Client Structure Inform	176
Chapter 8. CQS Diagnosis.	179
CQS Log Records	179
Printing CQS Log Records	181
DD Statements for CQS Diagnosis	181
Control Statements for CQS Diagnosis	181
Limiting Log Data to a Specified Time Range for CQS Diagnosis	182
Copying CQS Log Records for Diagnostics	182
Notices	185
Programming Interface Information	187
Trademarks.	188
Bibliography	189
IMS Version 9 Library	189
Index	191

Figures

1.	Client Systems, CQS, and the Coupling Facility	2
2.	Defining IMS Resources in the CFRM Policy	10
3.	Defining IMS Resources in the LOGR Policy.	11
4.	Defining IMS Resources in the SFM Policy	11
5.	Entry to Be Added to the z/OS Program Properties Table	13
6.	Specifying IMS and CQS Parameters	15
7.	Sample CQSIPxxx PROCLIB Member	16
8.	Sample CQSSLxxx PROCLIB Member.	18
9.	Sample CQSSGxxx PROCLIB Member	20
10.	System Checkpoint Data Set Example	25
11.	Structure Recovery Data Set Example	26
12.	RACF Commands for Authorizing CQS Registration	33
13.	RACF Commands to Authorize Connection to CQS Structures	34
14.	Display for Structure Full Threshold - Example 1	38
15.	Display for Structure Full Threshold - Example 2	38
16.	Display for Structure Full Threshold - Example 3	38
17.	Passing an Address for Register	73
18.	Passing a value for register	73
19.	Passing an Address for Symbol	73
20.	Passing a Value for Symbol	73
21.	Passing a Value for Symbol Value	74
22.	Passing an Equate for Symbol Value	74
23.	Coding CQSREAD with the OPTWORD1 parameter.	74
24.	STEPLIB DD Statement to Concatenate IMS.SDFSRESL	77
25.	Sample for CQSREAD Request	160
26.	JCL to Print CQS Log Records	181
27.	DD Card to Limit Log Records that are Printed	182
28.	DD Card to Add Local Date and Time.	182
29.	JCL to Copy CQS Records from a Specific Time Period	183

Tables

1.	How to Read Syntax Diagrams	xi
2.	Private Queue Types Managed by CQS	4
3.	CQS Init-Term User-Supplied Exit Routine Parameter List: CQS Initialization	49
4.	CQS Init-Term User-Supplied Exit Routine Parameter List: CQS Termination	49
5.	CQS Client Connection User-Supplied Exit Routine Parameter List: Client Connection	50
6.	CQS Client Connection User-Supplied Exit Routine Parameter List: Client Disconnect	50
7.	CQS Queue Overflow User-Supplied Exit Routine Parameter List	53
8.	CQS Structure Statistics User-Supplied Exit Routine Parameter List	54
9.	CQS Structure Process Statistics Record	55
10.	CQS Request Statistics Record	55
11.	Data Object Statistics Record	56
12.	Queue Name Statistics Record	57
13.	z/OS Request Statistics Record	57
14.	Structure Rebuild Statistics Record	58
15.	Structure Checkpoint Statistics Record	60
16.	Structure Checkpoint Statistics Entry	61
17.	CQS Structure Event User-Supplied Exit Routine Parameter List: Connect	63
18.	CQS Structure Event User-Supplied Exit Routine Parameter List: Checkpoint	64
19.	CQS Structure Event User-Supplied Exit Routine Parameter List: Rebuild	65
20.	CQS Structure Event User-Supplied Exit Routine Parameter List: Overflow	66
21.	CQS Structure Event User-Supplied Exit Routine Parameter List: Status Change	67
22.	CQS Statistics Header Data	68
23.	Sequence for CQS Requests	70
24.	Environment for CQS Requests (Excluding CQSREG and CQSDEREG) Using the Authorized Interface	71
25.	Environment for CQS Requests (Excluding CQSREG and CQSDEREG) Using the Non-Authorized Interface	71
26.	Environment for CQSREG and CQSDEREG Requests Using the Authorized Interface	72
27.	Environment for CQSREG and CQSDEREG Requests Using the Non-Authorized Interface	72
28.	Return and Reason Codes for Errors Detected by the CQS Interface	76
29.	CQSBRWSE Return and Reason Codes	86
30.	CQSCHKPT Return and Reason Codes	90
31.	CQSCONN Return and Reason Codes	95
32.	CQSDEL Return and Reason Codes	100
33.	CQSDEREG Return and Reason Codes	101
34.	CQSDISC Return and Reason Codes	105
35.	CQSINFRM Return and Reason Codes	110
36.	CQSMOVE Return and Reason Codes	113
37.	Actions Taken for Data Objects as a Result of Failures or Structure Activity	116
38.	CQSPUT Return and Reason Codes	120
39.	CQSQUERY Return and Reason Codes	129
40.	CQSREAD Return and Reason Codes	135
41.	CQSRECVR Return and Reason Codes	139
42.	CQSREG Return and Reason Codes	142
43.	UOW Status from the Client	145
44.	UOW Status from CQS	145
45.	CQSRSYNC Return and Reason Codes	147
46.	CQSSHUT Return and Reason Codes	150
47.	CQSUNLCK Return and Reason Codes	154
48.	CQSUPD Return and Reason Codes	159
49.	Client CQS Event Exit Routine Parameter List: CQS Restart Entry	166
50.	Client CQS Event Exit Routine Parameter List: CQS Abnormal Termination	167
51.	Client Structure Event Exit Routine Parameter List: Deferred Resync Complete	169

52.	Client Structure Event Routine Exit Parameter List: CQS Initiated Resync	170
53.	CQS Resync UOW Entry Parameters	170
54.	Client Structure Event Exit Routine Parameter List: Checkpoint	171
55.	Client Structure Event Exit Routine Parameter List: Structure Rebuild	172
56.	Client Structure Event Exit Routine Parameter List: Structure Rebuild Lost UOWs	173
57.	CQS Rebuild Lost UOW Entry Parameters	173
58.	Client Structure Event Exit Routine Parameter List: Structure Overflow	174
59.	Client Structure Event Exit Routine Parameter List: Structure Status Change	174
60.	Client Structure Inform Exit Routine Parameter List	176
61.	CQS Log Records	179

About This Book

This information is available in PDF and BookManager formats, and also as part of the IMS Version 9 QPP Information Center. 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. To get the most current versions of these books for the information center, go to the IMS V9 Vendor and Quality Partnership Program Library page at www6.software.ibm.com/dl/ims02/imsv9lib-p, where you can find updated plug-ins and instructions on how to install them in your IMS Version 9 QPP Information Center.

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.

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 189.

Related Reading: For definitions of terminology specific to CQS and used in this manual, see Chapter 1, “Introduction,” on page 1. Other terms are defined in the *IMS Version 9: Master Index and Glossary*.

How to Read Syntax Diagrams

Each syntax diagram in this book 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 1. How to Read Syntax Diagrams



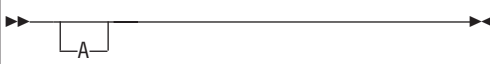




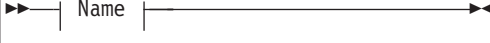
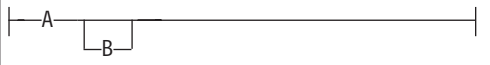
Convention	Meaning
	You must specify values A, B, and C. Required values are shown on the main path of a syntax diagram.
	You must specify value A, B, or C.

Table 1. How to Read Syntax Diagrams (continued)

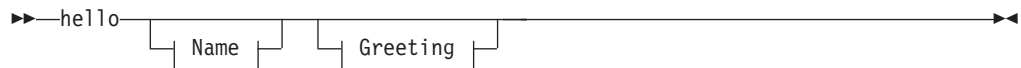
Convention	Meaning
	You have the option to specify value A. Optional values are shown below the main path of a syntax diagram.
	You have the option to specify A, B, C, or none of these values.
	You have the option to specify A, B, C, or none of these values. If you don't specify a value, A is the default.
	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.
	You have the option to specify value A multiple times. The separator in this example is optional.
 Name: 	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.
Punctuation marks and numbers	Enter punctuation marks (slashes, commas, periods, parentheses, quotation marks, equal signs) and numbers exactly as shown.
Uppercase values	Keywords, their allowable synonyms, and reserved parameters appear in uppercase letters for z/OS. Enter these values exactly as shown.
Lowercase values	Keywords, their allowable synonyms, and reserved parameters appear in lowercase letters for UNIX. Enter these values exactly as shown.
Lowercase values in italic (for example, <i>name</i>)	Supply your own text or value in place of the <i>name</i> variable.
b	A b symbol indicates one blank position.

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 *.

Syntax Diagram Example

Here is an example syntax diagram that describes the `hello` command.



Name:



Greeting:



Notes:

- 1 You can code up to three names.
- 2 Compose and add your own greeting (for example, how are you?).

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

```
hello
hello name
hello name, name
hello name, name, name
hello, your_greeting
hello name, your_greeting
hello name, name, your_greeting
hello name, name, name, 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 Library page at www.ibm.com/software/data/ims/library.html and click the Library Feedback link, 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 9

This edition contains editorial changes. In addition, the Glossary formerly located in Chapter 1, "Introduction," on page 1 has been removed. Refer to *IMS Version 9 Master Index and Glossary* for definitions of the terms previously defined in this book.

Changes to This Book for IMS Version 9

This edition is a draft version of the book intended for use during the Quality Partnership Program (QPP). Contents of this book are preliminary and under development.

New information on the following enhancements is included:

- Optional EMHQ Structure for Shared Queues: see "CQS Local Structure Definition PROCLIB Member (CQSSLxxx)" on page 17 and "CQS Global Structure Definition PROCLIB Member (CQSSGxxx)" on page 19.

In addition, the index has been expanded for enhanced retrievability.

Library Changes for IMS Version 9

Changes to the IMS Library for IMS Version 9 include the addition of new titles, the change of one title, and a major terminology change. Changes are indicated by a vertical bar (|) to the left of the changed text.

New and Revised Titles

The following list details the major changes to the IMS Version 9 library:

- *IMS Version 9: HALDB Online Reorganization Guide*
The library includes new information: *IMS Version 9: HALDB Online Reorganization Guide*. This information is available only in PDF and BookManager formats.
- *IMS Version 9: An Introduction to IMS*
The library includes new information: *IMS Version 9: An Introduction to IMS*.
- The information formerly titled *IMS Version 8: IMS Java User's Guide* is now titled *IMS Version 9: IMS Java Guide and Reference*.
- The library includes new information: *IMS Version 9: IMS Connect Guide and Reference*. This information is available only in PDF and BookManager formats.

Terminology Changes

IMS Version 9 introduces new terminology for IMS commands:

type-1 command

A command, generally preceded by a leading slash character, that can be entered from any valid IMS command source. In IMS Version 8, these commands were called *classic* commands.

type-2 command

A command that is entered only through the OM API. Type-2 commands

are more flexible and can have a broader scope than type-1 commands. In IMS Version 8, these commands were called *IMSplex* commands or *enhanced* commands.

Accessibility Enhancements

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products. 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 IMS user interfaces. Consult the documentation of the assistive technology products for specific information when you use assistive technology to access these interfaces.

Accessible Information

Online information for IMS Version 9 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 z/OS base product, and the BookManager Softcopy Reader (for workstations) is available on the IMS Licensed Product Kit (CD), which you can download from the Web 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 V1R1.0 TSO/E Primer*, the *z/OS V1R1.0 TSO/E User's Guide*, and the *z/OS V1R1.0 ISPF User's Guide, Volume 1*. 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

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:

- “Common Queue Server Overview”
- “Planning for CQS Hardware and Software Requirements” on page 6

This section contains General-Use Programming Interface information.

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 *z/OS MVS Setting Up a Sysplex* for complete details about setting up a sysplex.

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

CQS uses the 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 2 shows the communications and the relationship between clients, CQSs, and the coupling facility.

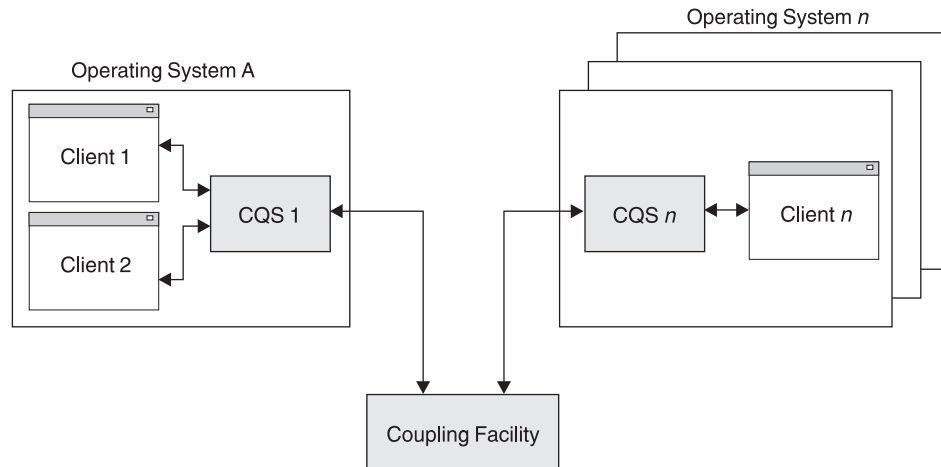


Figure 1. Client Systems, CQS, and the Coupling Facility

Related Reading: CQS requests are described in Chapter 6, “CQS Client Requests,” on page 79.

CQS Benefits

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
A z/OS coupling facility list structure that contains shared queues.
- Resource structure
A z/OS coupling facility list structure that contains uniquely named resources.
- Overflow structure
A z/OS coupling facility list structure that contains shared queues when the primary structure reaches an installation-specified overflow threshold. The overflow structure is optional.
- z/OS log stream
A shared z/OS 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.

- 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 z/OS 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 a z/OS 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 a collection of data objects, some of which might have the same name. Data objects that have the same name are considered to be on the same 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 2. Private Queue Types Managed by CQS

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

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.

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 37.

Structure Rebuild

Structure rebuild is a z/OS process that allows another instance of a structure to be allocated with the same name and contain data reconstructed from the initial structure instance. z/OS supports system-managed rebuild, in which case z/OS rebuilds the structure. z/OS also supports user-managed rebuild; the user rebuilds the structure. Structure rebuild can be initiated manually by using an operator command, or automatically by CQS or z/OS.

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.

Related Reading: For more information about rebuilding structures, see “Rebuilding Structures in CQS” on page 39.

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 27 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 z/OS 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 34 and “CQS Structure Event User-Supplied Exit Routine” on page 62 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 z/OS System Logger

CQS always uses the z/OS 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 28.

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 35 and “CQS Structure Statistics User-Supplied Exit Routine” on page 53 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 79.

Planning for CQS Hardware and Software Requirements

Refer to the *IMS Version 9: 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 *OS/390 Parallel 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

Chapter 2. CQS Definition and Tailoring

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

In this section:

- “CQS As Part of a Sysplex”
- “CQS and Defining z/OS Policies”
- “CQS’s Support of Multiple Clients” on page 11
- “Preparing to Start CQS” on page 12
- “Updating z/OS Program Properties Table for CQS” on page 13
- “CQS Execution Parameters” on page 14
- “CQS Initialization Parameters PROCLIB Member (CQSIPxxx)” on page 16
- “CQS Local Structure Definition PROCLIB Member (CQSSLxxx)” on page 17
- “CQS Global Structure Definition PROCLIB Member (CQSSGxxx)” on page 19
- “CQS Execution Data Sets” on page 24
- “CQS System Checkpoint Data Set” on page 24
- “CQS Structure Recovery Data Sets” on page 25

This section 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 policies:

- CQS for managing shared queues and resources
- CQS clients
- z/OS Operating System
- Signaling paths between systems
- Sysplex Couple Data Set that contains z/OS 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

“CQS and Defining z/OS Policies,” provides guidance on how to define the coupling facility resource management (CFRM), SFM, and LOGR policies for a typical IMS sysplex using CQS. “CQS Execution Data Sets” on page 24 and “Using the z/OS Automatic Restart Manager with CQS” on page 31 provide guidance on using the Automatic Restart Manager with CQS.

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

CQS and Defining z/OS Policies

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

Definition: A policy is a set of rules and actions that systems in a sysplex must follow when using certain z/OS services. A policy allows z/OS 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 z/OS for systems management in a sysplex environment and are required for the CQS:

- The automatic restart management (ARM) policy defines how z/OS is to manage restarts for specific z/OS 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 z/OS 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 12.

Users who do not intend to use Shared Expedited Message Handler (Shared EMH) processing in a sysplex can disable the EMH queue (EMHQ). In a CQS environment, you must modify the CQSSLxxx and CQSSGxxx PROCLIB members to disable EMHQ usage. See “CQS Local Structure Definition PROCLIB Member (CQSSLxxx)” on page 17 and “CQS Global Structure Definition PROCLIB Member (CQSSGxxx)” on page 19 for additional information.

- 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 65 272 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 10 shows an example of a CFRM policy that defines the following IMS resources:

- EMHQ primary structure
- EMHQ overflow structure
- EMHQ log structure
-
- Message queue (MSGQ) primary structure
- MSGQ overflow structure
- MSGQ log structure
- Resource structure

Figure 3 on page 11 shows an example of how the LOGR policy can be defined.

Figure 4 on page 11 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:

- *z/OS MVS Setting Up a Sysplex* for the CFRM, SFM, LOGR, and ARM policies
- *z/OS MVS Programming: Assembler Services Guide* for the LOGR policy.

The example in Figure 2 on page 10 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
//STEPLIB DD DSN=SYS1.MIGLIB,DISP=SHR
//SYSPRINT 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(QMSGIMS010FLW)
        SIZE(8000)
        PREFLIST(CF01,CF02)
        REBUILDPERCENT(1)

STRUCTURE NAME(QEMHIMS01)
        SIZE(16000)
        INITSIZE(10000)
        PREFLIST(CF01,CF02)
        REBUILDPERCENT(1)

STRUCTURE NAME(QEMHIMS010FLW)
        SIZE(8000)
        PREFLIST(CF01,CF02)
        REBUILDPERCENT(1)

```

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

```

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

STRUCTURE NAME(MVSLOGQEMH01)
  SIZE(4000)
  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.

```

DATA TYPE(LOGR)
DEFINE STRUCTURE NAME(MVSLOGQMSG01)
  LOGSNUM(1)
  AVGBUFSIZE(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.

```

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 z/OS 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 34.

Preparing to Start CQS

Because they are a part of IMS, CQS and Base Primitive Environment (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 z/OS 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=SFM,POLNAME=SFMPOL
```
5. Create the CQS and BPE PROCLIB members. For information on creating BPE PROCLIB members, see *IMS Version 9: Base Primitive Environment 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 z/OS program properties table.

10. Plan security.

You 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 z/OS policies, see “CQS and Defining z/OS Policies” on page 7.
- For more information on the CQS initialization parameters, see “CQS Initialization Parameters PROCLIB Member (CQSIPxxx)” on page 16.
- For more information on the CQS user exit routines, see Chapter 4, “CQS User-Supplied Exit Routines,” on page 47.
- For more information on BPE user exit routines, see the *IMS Version 9: Base Primitive Environment Guide and Reference*.

Updating z/OS Program Properties Table for CQS

You must add an entry in the z/OS 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:

```

PPT PGMNAME(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 z/OS Program Properties Table

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

Related Reading: For additional reading about updating the program properties table, see *z/OS MVS Initialization and Tuning Reference*.

CQS Execution Parameters

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 z/OS 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 z/OS, you must specify the ARMRESTART option of either the z/OS CANCEL or FORCE command.

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

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

BPECFG=

Specifies the 8-character name of the BPE configuration PROCLIB member. You can specify this parameter only as an execution parameter.

CQSGROUP=

Specifies a 1- to 5-character identifier. CQS concatenates this identifier to the characters CQS to create the cross-system coupling facility (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 3-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 1 to 4 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 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 3-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 3-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 9: 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.

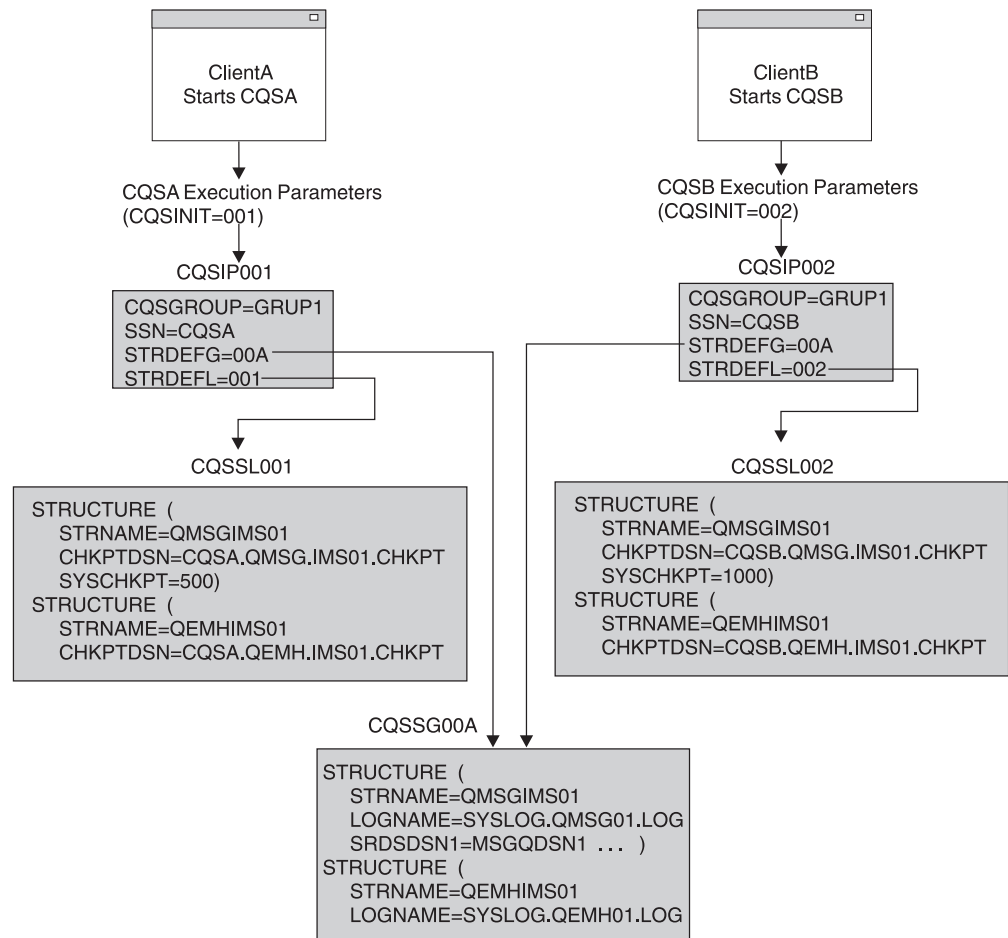


Figure 6. Specifying IMS and CQS Parameters

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 logical record length (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.

```
/*This is an example of a comment within a statement*/
```
- 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 z/OS 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 z/OS, you must specify the ARMRESTART option of either the z/OS CANCEL or FORCE command.

Related Reading: For information on the CANCEL and FORCE commands, see *z/OS 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 1- to 5-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 1- to 5-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 CSLSIxxx, CSLOIxxx, CSLRIxxx and DFSCGxxx PROCLIB members.

SSN=

Specifies the name for the CQS address space. The value must be 1 to 4 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 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 3-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 3-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.

If the STRUCTURE statement for an EMHQ structure is deleted from the CQSSLxxx PROCLIB member, resources for the EMHQ structure and its associated CQS data sets are not allocated.

```
*****
* 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 2 147 483 647. Each CQS address space that is connected to a 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 queue structures, resource structures, or a combination of both queue structures and resource structures. Each CQS sharing a structure must point to a CQSSGxxx member containing identical structure definition parameters.

Recommendations:

- Point all CQSSs 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 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 CQSSs 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.

If the STRUCTURE statement for an EMHQ structure is deleted from the CQSSGxxx PROCLIB member, resources for the EMHQ structure are not allocated. These resources include the EMHQ structure's associated overflow structure, structure recovery data sets, and CQS log.

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=QMSGIMS010FLW,
  SRSDSN1=CQS.QMSG.IMS01.SRDS1
  SRSDSN2=CQS.QMSG.IMS01.SRDS2,
  LOGNAME=SYSLOG.QMSG01.LOG
  OBJAVGSZ=1024)

*-----*
* DEFINITION FOR IMS EMH QUEUE STRUCTURES          *
*-----*
STRUCTURE (
  STRNAME=QEMHIMS01,
  OVFLWSTR=QEMHIMS010FLW,
  SRSDSN1=CQS.QEMH.IMS01.SRDS1,
  SRSDSN2=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=*strname*, SRSDSN1=*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 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 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 z/OS log stream that CQS uses to record all information related to the structure. The installation must have previously defined this name to the z/OS system logger.

All CQS address spaces that connect to a 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 61312 bytes or from 1K to 59K. 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.

$$(134 + 1066 + 3200 + 172 + 345)/5 = 983$$

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 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 structure 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 queue structure maintains a system checkpoint data set for each structure pair. Neither CQS address spaces nor queue 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 *z/OS 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 *z/OS 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 SRSDSDSN1= and the SRSDSDSN2= 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 32 768; 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

Chapter 3. CQS Administration

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

In this section:

- “Starting CQS”
- “Recording Information Necessary for Starting CQS” on page 28
- “Restarting CQS” on page 30
- “Restarting CQS Structures” on page 28
- “Establishing Client Connection to CQS During Failed Client Takeover” on page 32
- “Authorizing Access To CQS” on page 33
- “Using Structure Alter for CQS” on page 34
- “Using CQS System Checkpoint” on page 34
- “Using CQS Structure Checkpoint” on page 35
- “Preventing CQS Structure Full” on page 37
- “Rebuilding Structures in CQS” on page 39
- “Deleting a Structure When CQS Is Not Connected” on page 44
- “Shutting Down CQS” on page 44

This section contains Product-sensitive Programming Interface information.

Starting CQS

You can start CQS in one of three ways:

- As a z/OS task, using the z/OS START command
- As a z/OS 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 9: Installation Volume 2: System Definition and Tailoring*.
- For information on defining z/OS policies, see “CQS and Defining z/OS Policies” on page 7.
- For information on the CQS initialization parameters, see “CQS Initialization Parameters PROCLIB Member (CQSIPxxx)” on page 16.

Recording Information Necessary for Starting CQS

CQS uses the z/OS 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 z/OS 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 a z/OS system log, you must first define the log stream and associated resources to z/OS.

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 *z/OS MVS Setting Up a Sysplex*.

For more information on the File Select and Formatting Print utility, see “Printing CQS Log Records” on page 181.

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 z/OS 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.

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 z/OS 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 write-to-operator-with-reply (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 41.

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 check pointed 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 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 35 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 CQs 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 z/OS Automatic Restart Manager with CQS

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

Recommendation: Register with the z/OS ARM regardless of the types of structures CQS is using.

To enable the ARM, you can specify ARMRST=Y in one of two ways:

- In the CQSIPxxx PROCLIB member
- As an execution parameter

An abend table exists in module CQSARM10. The table lists the CQS abends for which the ARM 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 ARM 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 ARM is enabled, CQS registers to ARM with an ARM element name of CQS + *cqsssn* + CQS. Use this ARM element name in the ARM policy to define the ARM policy for CQS.

cqsssn is the CQS name defined either as a CQS execute parameter, or in the CQSIPxxx PROCLIB member with the SSN= parameter. For example, if SSN=CQSA, then the ARM element name is CQSCQSACQS.

Related Reading: For more information on using the z/OS ARM with CQS, see “CQS Execution Parameters” on page 14. See *z/OS MVS Setting Up a Sysplex*, GC28-1779 for OS/390® or SA22-7625 for z/OS, for details on z/OS ARM policy.

Restarting CQS After CQS Resource Cleanup Failures

If CQS abends and you receive message CQS0102E with module CQSRS000, 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 CQSSs are running, either immediately restart the terminated CQS or initiate a structure checkpoint on one of the surviving CQSSs. For more information, see the /CQCHKPT SHAREDQ command in *IMS Version 9: 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 CQSIPxxx 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 *z/OS MVS Programming: Sysplex Services Guide*.

For more information on defining structure names, see “CQS Global Structure Definition PROCLIB Member (CQSSGxxx)” on page 19 and “CQS Local Structure Definition PROCLIB Member (CQSSLxxx)” on page 17. 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 a z/OS 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 12.

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 a z/OS 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 z/OS, if the coupling facility storage becomes constrained. Be careful when enabling automatic structure alter for queue structures. If z/OS 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 z/OS 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 CQSCHKPT 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 DSNAME 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 CQSSs, CQS restart time is affected by the number of log records written by the multiple CQSSs, 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 SRDS. For nonrecoverable data objects, the queue name, and UOW are copied, 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:

- When the z/OS log becomes full or approaches full.
- After a successful structure recovery.
- After a successful overflow threshold process.
- When a client issues the CQSCHKPT FUNC=CHKPTSTR request.
- When the client is IMS, and you enter the /CQCHKPT SHAREDQ command.

- During CQS normal termination when the client requests it on the CQSDISC 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 or queue structure becomes full, CQS issues message CQS0205E. There are two ways to prevent a structure full condition:

- CQS structure overflow function for queue structures
- z/OS 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 z/OS 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, z/OS monitors structure usage. When the number of entries or elements in use reaches the structure full threshold, z/OS issues a highlighted IXC585E message to warn the system programmer that a structure full condition is imminent. If automatic altering is enabled, z/OS automatically initiates a structure alter to increase the structure size or change the element to entry ratio.

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 FULLTHRESHOLD(0) to disable structure full monitoring. When the structure usage goes below the threshold, z/OS 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: IMSR SRC01
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

Figure 15 shows the IXC585E message, indicating the structure is full because all of the entries are in use:

```
*IXC585E STRUCTURE IMSR SRC01 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 shows the IXC586I message:

```
IXC586I STRUCTURE IMSR SRC01 IN COUPLING FACILITY LF03, 772
PHYSICAL STRUCTURE VERSION B4704775 92D95302,
IS NOW BELOW STRUCTURE FULL MONITORING THRESHOLD.
```

Figure 16. Display for Structure Full Threshold - Example 3

Related Reading: For more details on structure full monitoring and the FULLTHRESHOLD and ALLOWAUTOALT keywords in the CFRM policy, see *z/OS MVS Setting Up a Sysplex*.

Using Structure Full Monitoring with CQS Structure Overflow

You can use the structure full monitoring function (a z/OS 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 z/OS. If the overflow threshold is close to the structure full threshold and automatic altering is enabled, CQS and z/OS might 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 z/OS detects the structure full threshold has been reached, z/OS 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 z/OS-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, z/OS 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 19.
- For detailed information about the Queue Overflow User-Supplied exit routine, see “Queue Overflow User-Supplied Exit Routine for CQS” on page 51.
- For detailed information about the CQSPUT request, see “CQSPUT Request” on page 114.

Rebuilding Structures in CQS

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

z/OS System-Managed Rebuild and CQS

System-managed rebuild is a z/OS process by which z/OS rebuilds the structure. z/OS 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, z/OS 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 job control language (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 z/OS 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 CQs 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 z/OS and CQS

A structure rebuild can be initiated by a z/OS operator, by CQS, or by z/OS:

- A z/OS operator can initiate a structure rebuild to copy or recover queues using the following command:

```
SETXCF START,REBUILD,STRNAME=strname,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 event notification facility (ENF) 35 event processing.
- z/OS 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, z/OS 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 z/OS 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.

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=NO 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.

z/OS Structure Duplexing for CQS

Structure duplexing is an optional z/OS-managed process for failure recovery of queue structures and resource structures. In this process, z/OS 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, z/OS 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 z/OS 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, z/OS 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 z/OS 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 z/OS 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 z/OS 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 you enable z/OS 1.2 duplexing, you cannot return to downlevel 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.

After an uplevel 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 z/OS internally can initiate the duplexing rebuild. z/OS 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), z/OS 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 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.

Once duplexing is established, the structure remains in that state indefinitely. Duplexing can be stopped internally by z/OS 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 CQSSs 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 z/OS STOP command to shut down CQS.

Related Reading: See “Shutting Down CQS” on page 78 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 z/OS 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 section the term “user exit routine” means “user-supplied exit routine.”

This section describes the following CQS user exit routines:

“General User-Supplied Exit Routine Interface Information for CQS”

“CQS Initialization-Termination User-Supplied Exit Routine” on page 48

“CQS Client Connection User-Supplied Exit Routine” on page 49

“Queue Overflow User-Supplied Exit Routine for CQS” on page 51

“CQS Structure Statistics User-Supplied Exit Routine” on page 53

“CQS Structure Event User-Supplied Exit Routine” on page 62

“CQS Statistics Available through the BPE Statistics User Exit” on page 67

This section 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.

A list of the user exit routines and their functions follows:

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 67

General User-Supplied Exit Routine Interface Information for CQS

CQS uses BPE services to call and manage its user exit routines. BPE allows you to externally specify the user exit routine modules to be called for a particular exit routine type using 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
- Static work areas for the routines
- Dynamic work areas for the routines

- Callable services for the routines
- A recovery environment to protect against abends in the user exit routines

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 z/OS. If you write an exit routine in a high level language, and that routine is executing in the Language Environment for z/OS, you might have abends or performance problems. Language Environment for z/OS 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 9: Base Primitive Environment Guide and Reference*.
- For complete information about BPE interfaces and services that are available to user exits, see *IMS Version 9: 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

Register	Contents
1	Address of Standard BPE user exit parameter list (mapped by the BPEUXPL macro). See <i>IMS Version 9: Base Primitive Environment Guide and Reference</i> 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

Register	Contents
-----------------	-----------------

15 Return code
0 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

Field Name	Offset	Length	Field Usage	Description
ITXPVSN	X'00'	X'04'	Input	Parameter List Version Number (00000001)
ITXFUNC	X'04'	X'04'	Input	Function code 1 CQS Initialization (ITXFINIT)
ITXCQSID	X'08'	X'08'	Input	CQS identifier
ITXCQSVN	X'10'	X'04'	Input	CQS version number

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

Field Name	Offset	Length	Field Usage	Description
ITXPVSN	X'00'	X'04'	Input	Parameter List Version Number (00000001)
ITXFUNC	X'04'	X'04'	Input	Function code 2 CQS Normal Termination (ITXFNTRM)
ITXCQSID	X'08'	X'08'	Input	CQS identifier
ITXCQSVN	X'10'	X'04'	Input	CQS version number

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.

Contents of Registers on Entry

Register	Contents
1	Address of Standard BPE user exit parameter list (mapped by the

BPEUXPL macro). See *IMS Version 9: Base Primitive Environment 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

Register	Contents
15	Return code
0	Always set this to zero.

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_EXITPLP 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

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

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

Field Name	Offset	Length	Field Usage	Description
CCXPVSN	X'00'	X'04'	Input	Parameter List Version Number (00000001).
CCXFUNC	X'04'	X'04'	Input	Function code
				2 Client Disconnect (CCXFDISC).
CCXCQSID	X'08'	X'08'	Input	CQS identifier.
CCXCQSVN	X'10'	X'04'	Input	CQS version number.
CCXCLNNM	X'14'	X'08'	Input	Client name.
CCXDFLG1	X'1C'	X'01'	Input	Flag byte indicates whether the client disconnect is abnormal
				X'80' Client disconnect is abnormal (CCXDABND).

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

Field Name	Offset	Length	Field Usage	Description
N/A	X'1D'	X'03'		Reserved.
CCXDSNUM	X'20'	X'04'	Input	Number of structure name entries in the list.
CCXDSENL	X'24'	X'04'	Input	Length of each structure name list entry.
CCXDSLST	X'28'	X'04'	Input	Address of first structure name entry. Each entry contains the 16-byte name of a structure that the client disconnected from.

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.

Within the Standard BPE user exit parameter list is the field UXPL_CALLNEXT, 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

Register	Contents
1	Address of Standard BPE user exit parameter list (mapped by the BPEUXPL macro). See <i>IMS Version 9: Base Primitive Environment Guide and Reference</i> 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

Register	Contents
15	Return code <ul style="list-style-type: none"> 0 Allow queue to be moved to overflow structure. 4 Do not move queue to overflow structure; select another candidate. <p>Attention: This return code is ignored unless the exit routine is the last overflow user exit called for the queue name.</p> <p>An exit routine is considered the last one called when either of the following are true:</p> <ol style="list-style-type: none"> 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 on page 53.

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

Field Name	Offset	Length	Field Usage	Description
QOXPVSN	X'00'	X'04'	Input	Parameter List Version Number (00000001).
QOXFUNC	X'04'	X'04'	Input	Function code 1 Queue Name Selection (QOXFQOFL).
QOXQOFL1	X'08'	X'01'	Input	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 (QOXQ11ST)
N/A	X'09'	X'03'		Reserved.
QOXCQSID	X'0C'	X'08'	Input	CQS identifier.
QOXCQSVN	X'14'	X'04'	Input	CQS version number.
QOXSTRNM	X'18'	X'10'	Input	Structure Name.
QOXQNAME	X'28'	X'10'	Input	Queue name selected for overflow processing.
QOXDOBJN	X'38'	X'04'	Input	Number of data objects on the selected queue name.

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 9: 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

Register	Contents
1	Address of Standard BPE user exit parameter list (mapped by the BPEUXPL macro). See <i>IMS Version 9: Base Primitive Environment Guide and Reference</i> 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

Register	Contents
15	Return code
0	Always set this to zero.

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 CQSSTATX macro). The parameters are described in Table 8.

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

Field Name	Offset	Length	Field Usage	Description
SAXPVSN	X'00'	X'04'	Input	Parameter List Version Number (00000001).
SAXFUNC	X'04'	X'04'	Input	Function code 1 System Checkpoint (SAXFCSYS).
SAXCQSID	X'08'	X'08'	Input	CQS identifier.
SAXCQSVN	X'10'	X'04'	Input	CQS version number.
SAXSTRNM	X'14'	X'10'	Input	Structure name.
SAXSSTT1	X'24'	X'04'	Input	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 55 for a description of the process statistics record.
SAXSSTT2	X'28'	X'04'	Input	Address of CQS request statistics record for activity performed for CQS requests for this structure for all clients since restart or the last successful structure checkpoint (mapped by the CQSSSTT2 macro). See Table 10 on page 55 for a description of the request statistics record.
SAXSSTT3	X'2C'	X'04'	Input	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 56 for a description of the object statistics record.
SAXSSTT4	X'30'	X'04'	Input	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 57 for a description of the queue name statistics record.
SAXSSTT5	X'34'	X'04'	Input	Address of z/OS 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 57 for a description of the z/OS request statistics record.

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

Field Name	Offset	Length	Field Usage	Description
SAXSSTT6	X'38'	X'04'	Input	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 58 for a description of the rebuild statistics record.
SAXSSTT7	X'3C'	X'04'	Input	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 60 for a description of the structure checkpoint statistics record.

CQS Structure Process Statistics Record

Table 9 describes the CQS Structure Statistics user exit routine structure process statistics record.

Table 9. CQS Structure Process Statistics Record

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

CQS Request Statistics Record

Table 10 describes the Structure Statistics user exit routine CQS request statistics record.

Table 10. CQS Request Statistics Record

Field Name	Offset	Length	Field Usage	Description
SS2ID	X'00'	X'08'	Input	Eyecatcher CQSSSTT2
SS2LN	X'08'	X'04'	Input	Length of valid data
SS2PVSN	X'0C'	X'04'	Input	Parameter List Version Number (00000002)
SS2BRWSE	X'10'	X'04'	Input	Number of CQSBWSE requests for this structure
SS2CHKPT	X'14'	X'04'	Input	Number of CQSCHKPT requests for this structure
SS2CONN	X'18'	X'04'	Input	Number of CQSCONN requests for this structure
SS2DEL	X'1C'	X'04'	Input	Number of CQSDDEL requests for this structure
SS2DISC	X'20'	X'04'	Input	Number of CQSDISC requests for this structure
SS2INFRM	X'24'	X'04'	Input	Number of CQSINFRM requests for this structure
SS2MOVE	X'28'	X'04'	Input	Number of CQSMOVE requests for this structure
SS2PUT	X'2C'	X'04'	Input	Number of CQSPUT requests for this structure
SS2QUERY	X'30'	X'04'	Input	Number of CQSQUERY requests for this structure

Table 10. CQS Request Statistics Record (continued)

Field Name	Offset	Length	Field Usage	Description
SS2READ	X'34'	X'04'	Input	Number of CQSREAD requests for this structure
SS2RECVR	X'38'	X'04'	Input	Number of CQSRECVR requests for this structure
SS2RSYNC	X'3C'	X'04'	Input	Number of CQSRSYNC requests for this structure
SS2UNLCK	X'40'	X'04'	Input	Number of CQSUNLCK requests for this structure
SS2UPD	X'44'	X'04'	Input	Number of CQSUPD requests for this structure

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

Field Name	Offset	Length	Field Usage	Description
SS3ID	X'00'	X'08'	Input	Eyecatcher CQSSSTT3.
SS3LN	X'08'	X'04'	Input	Length of valid data.
SS3PVSN	X'0C'	X'04'	Input	Parameter List Version Number (00000002).
SS3PTOBJ	X'10'	X'04'	Input	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.
SS3PTCMT	X'14'	X'04'	Input	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.
SS3PTNRO	X'18'	X'04'	Input	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.
SS3RDOBJ	X'1C'	X'04'	Input	Number of data objects read from the structure.
SS3MVOBJ	X'20'	X'04'	Input	Number of data objects moved from one queue to another on the structure.
SS3ULOBJ	X'24'	X'04'	Input	Number of data objects unlocked on the structure.
SS3ENTAL	X'30'	X'04'	Input	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.
SS3ENTIN	X'34'	X'04'	Input	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.
SS3ENTHI	X'38'	X'04'	Input	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.
SS3ENTTM	X'3C'	X'08'	Input	Timestamp representing the time the data entry high water mark was reached for the structure (in STCK format).
SS3ELMAL	X'44'	X'04'	Input	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.
SS3ELMIN	X'48'	X'04'	Input	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.

Table 11. Data Object Statistics Record (continued)

Field Name	Offset	Length	Field Usage	Description
SS3ELMHI	X'4C'	X'04'	Input	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.
SS3ELMTM	X'50'	X'04'	Input	Timestamp representing the time the data element high water mark was reached for the structure (in STCK format).
Reserved	X'58'	X'04'	Input	
Reserved	X'5C'	X'04'	Input	

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

Field Name	Offset	Length	Field Usage	Description
SS4ID	X'00'	X'08'	Input	Eyecatcher CQSSSTT4
SS4LN	X'08'	X'04'	Input	Length of valid data
SS4PVSN	X'0C'	X'04'	Input	Parameter List Version Number (00000001)
SS4INFQN	X'10'	X'04'	Input	Number of queue names for which an inform was performed
SS4UNFQN	X'14'	X'04'	Input	Number of queue names for which an uninform was performed
SS4NFYQN	X'18'	X'04'	Input	Number of queue name notifications (when a queue goes from empty to non-empty)

z/OS Request Statistics Record for CQS

Table 13 describes the Structure Statistics user exit routine z/OS request statistics record.

Table 13. z/OS Request Statistics Record

Field Name	Offset	Length	Field Usage	Description
SS5ID	X'00'	X'08'	Input	Eyecatcher CQSSSTT5.
SS5LN	X'08'	X'04'	Input	Length of valid data.
SS5PVSN	X'0C'	X'04'	Input	Parameter List Version Number (00000002).
SS5IXGWR	X'10'	X'04'	Input	Number of IXGWRITE requests for the structure. This represents the number of log records written during processing on the structure.
SS5IXGBR	X'14'	X'04'	Input	Number of IXGBRWSE requests for the structure.
SS5IXLDQ	X'18'	X'04'	Input	Number of IXLLIST DEQ_EVENTQ requests for the structure.
SS5IXLWR	X'1C'	X'04'	Input	Number of IXLLIST WRITE requests for the structure.
SS5IXLRD	X'20'	X'04'	Input	Number of IXLLIST READ requests for the structure.
SS5IXLMV	X'24'	X'04'	Input	Number of IXLLIST MOVE requests for the structure.
SS5IXLDL	X'28'	X'04'	Input	Number of IXLLIST DELETE requests for the structure.
SS5IXLMG	X'2C'	X'04'	Input	Number of IXLMG requests for the structure.
SS5IXLUS	X'30'	X'04'	Input	Number of IXLUSYNC requests for the structure.
SS5IXEWR	X'34'	X'04'	Input	Number of IXLLSTE WRITE requests for the structure.

Table 13. z/OS Request Statistics Record (continued)

Field Name	Offset	Length	Field Usage	Description
SS5IXERD	X'38'	X'04'	Input	Number of IXLLSTE READ requests for the structure.
SS5IXMRL	X'3C'	X'04'	Input	Number of IXLLSTM READ_LIST requests for the structure.
SS5IXEDL	X'40'	X'04'	Input	Number of IXLLSTE DELETE requests for the structure.
SS5IXMDL	X'44'	X'04'	Input	Number of IXLLSTM DELETE_ENTRYLIST requests for the structure.

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

Field Name	Offset	Length	Field Usage	Description
SS6ID	X'00'	X'08'	Input	Eyecatcher CQSSSTT6.
SS6LN	X'08'	X'04'	Input	Length of valid data.
SS6PVSN	X'0C'	X'04'	Input	Parameter List Version Number (00000003).
SS6ELMIO	X'10'	X'04'	Input	Data elements in use on old structure.
SS6ELMAO	X'14'	X'04'	Input	Data elements allocated on old structure.
SS6ENTIO	X'18'	X'04'	Input	Data entries in use on old structure (data object count).
SS6ENTAO	X'1C'	X'04'	Input	Data entries allocated on old structure.
SS6MCIO	X'20'	X'04'	Input	Event monitoring controls (EMCs) in use on old structure (active informs).
SS6EMCAO	X'24'	X'04'	Input	EMCs in use on old structure (active informs).
SS6SIZEO	X'28'	X'04'	Input	Old structure size in 4K blocks.
SS6CFTO	X'2C'	X'04'	Input	Old CF total space in 4K blocks.
SS6CFFO	X'30'	X'04'	Input	Old CF free space in 4K blocks.
SS6CFNMO	X'34'	X'08'	Input	Old CF name in which structure was allocated before rebuild.
	X'3C'	X'04'		Unused.
SS6ELMIN	X'40'	X'04'	Input	Data elements in use on new structure.
SS6ELMAN	X'44'	X'04'	Input	Data elements allocated on new structure.
SS6ENTIN	X'48'	X'04'	Input	Data entries in use on new structure (data object count).
SS6ENTAN	X'4C'	X'04'	Input	Data entries allocated on new structure.
SS6EMCIN	X'50'	X'04'	Input	EMCs in use on new structure (active informs).
SS6EMCAN	X'54'	X'04'	Input	EMCs in use on new structure (active informs).
SS6SIZEN	X'58'	X'04'	Input	New structure size in 4K blocks.
SS6CFTN	X'5C'	X'04'	Input	New CF total space in 4K blocks.
SS6CFFN	X'60'	X'04'	Input	New CF free space in 4K blocks.
SS6CFNMN	X'64'	X'08'	Input	New CF name in which structure is allocated after rebuild.
	X'6C'	X'04'		Unused.
SS6RBTIM	X'70'	X'08'	Input	Rebuild timestamp (STCK).
SS6POPCT	X'78'	X'04'	Input	Repopulation from SRDS count (RCVRY) or objects copied count (COPY).
SS6MVQCT	X'7C'	X'04'	Input	Entries moved to moveq during phase 2 count.

Table 14. Structure Rebuild Statistics Record (continued)

Field Name	Offset	Length	Field Usage	Description
SS6PUTCT	X'80'	X'04'	Input	Entries written during phase 3 count.
SS6MOVCT	X'84'	X'04'	Input	Entries moved during phase 3 count.
SS6OBJCT	X'88'	X'04'	Input	Data objects affected by recovery count (recoverable and nonrecoverable).
SS6UOWCT	X'8C'	X'04'	Input	UOWs affected by recovery count (recoverable and nonrecoverable).
SS6FLAG1	X'90'	X'01'	Input	Flag byte. X'80' These statistics are for the last rebuild performed for the structure.
SS6FLAG2	X'91'	X'01'	Input	Rebuild flag. Indicates the last rebuild or duplexing rebuild event received that updated these rebuild statistics: 1 Structure rebuild statistics. 2 Duplexing started statistics. 3 Duplexing ended statistics and z/OS switched to simplex structure (either old or new structure).
	X'91'	X'03'		Unused.
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.				
SS6LGTYP	X'94'	X'01'	Input	Log record type of log record being processed when rebuild failure occurred.
SS6LGSUB	X'95'	X'01'	Input	Log record subtype of log record being processed when rebuild failure occurred.
SS6STYPE	X'96'	X'01'	Input	Structure type of log record being processed when rebuild failure occurred.
	X'97'	X'01'		Unused.
SS6LGTIM	X'98'	X'08'	Input	Log record time stamp of log record being processed when rebuild failure occurred.
SS6CQSID	X'A0'	X'08'	Input	CQS ID associated with log record being processed when rebuild failure occurred.
SS6CLNTN	X'A8'	X'08'	Input	Client name associated with log record being processed when rebuild failure occurred.
SS6SRCQ	X'B0'	X'10'	Input	Source client or private queue name associated with log record being processed when rebuild failure occurred.
SS6DSTQ	X'C0'	X'10'	Input	Destination queue name associated with log record being processed when rebuild failure occurred.
SS6UOW	X'B0'	X'20'	Input	UOW associated with log record being processed when rebuild failure occurred.
SS6UNIQ1	X'F0'	X'04'	Input	Information unique to log record or rebuild data object entry when rebuild failure occurred.
SS6UNIQ2	X'F4'	X'04'	Input	Information unique to log record or rebuild data object entry when rebuild failure occurred.
SS6UNIQ3	X'F8'	X'04'	Input	Information unique to log record or rebuild data object entry when rebuild failure occurred.
The following fields apply to rebuild failures that occurred while rebuild was processing an IXL request to access the structure.				
SS6IXLMC	X'FC'	X'01'	Input	IXL macro that failed and caused rebuild to fail. See CQSTRACE macro for IXL macro type.
SS6IXLRQ	X'FD'	X'01'	Input	IXL request that failed and caused the rebuild to fail.
	X'FE'	X'02'		Unused.

Table 14. Structure Rebuild Statistics Record (continued)

Field Name	Offset	Length	Field Usage	Description
SS6IXLRC	X'100'	X'04'	Input	IXL return code returned by IXL request that caused rebuild to fail.
SS6IXLRN	X'104'	X'04'	Input	IXL reason code returned by IXL request that caused rebuild to fail.
SS6SRVRC	X'108'	X'04'	Input	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.
	X'10C'	X'04'		Unused.
SS6VRSNO	X'110'	X'08'	Input	Old structure version (rebuild) or primary structure version (duplexing rebuild).
SS6VRSNN	X'118'	X'08'	Input	New structure version (rebuild) or secondary structure version (duplexing rebuild).
SS6CFLVO	X'120'	X'04'	Input	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.
SS6CFLVN	X'124'	X'04'	Input	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.
SS6CFNMS	X'128'	X'04'	Input	CF name in which simplex structure is located (z/OS switched to simplex structure).
SS6VALFL	X'12C'	X'02'	Input	Validity flags (EPELSSCVALIDITYFLAGS).
	X'12E'	X'02'	Input	Not used
SS6DUPST	X'130'	X'08'	Input	Last duplexing rebuild start time (STCK). The last duplexing rebuild for this structure was initiated at this time.
SS6DUPET	X'138'	X'08'	Input	Last duplexing rebuild end time (STCK). The last duplexing rebuild stopped for this structure occurred at this time.
SS6UNAVT	X'140'	X'08'	Input	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.
SS6AVT	X'148'	X'08'	Input	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.
	X'150'	X'38'	Input	Unused

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 describes the Structure Statistics user exit routine structure checkpoint statistics record.

Table 15. Structure Checkpoint Statistics Record

Field Name	Offset	Length	Field Usage	Description
SS7ID	X'00'	X'08'	Input	Eyecatcher CQSSSTT7.
SS7LN	X'08'	X'04'	Input	Length of valid data.

Table 15. Structure Checkpoint Statistics Record (continued)

Field Name	Offset	Length	Field Usage	Description
SS7PVSN	X'0C'	X'04'	Input	Parameter List Version Number.
SS7FLAG1	X'10'	X'01'	Input	Flag byte.
				X'80' These statistics are from last attempted structure checkpoint taken for the structure.
				X'40' Structure Checkpoint is in progress.
	X'11'	X'03'		Unused.
SS7ENCNT	X'14'	X'04'	Input	Number of structure checkpoint statistics entries in record.
SS7ENLEN	X'18'	X'04'	Input	Length of structure checkpoint statistics entry
SS7CUR	X'1C'	X'04'	Input	Offset to current structure checkpoint statistics entry.
SS7STATS	X'20'	X''		Start of structure checkpoint statistics entries. See Table 16 on page 61 for a description of the structure checkpoint statistics entry.

Structure Checkpoint Statistics Gathered by 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

Field Name	Offset	Length	Field Usage	Description
SS7RETC	X'00'	X'08'	Input	Return Code for this Structure Checkpoint
SS7QSCB	X'08'	X'08'	Input	Structure quiesce start time in STCK format
SS7QSCE	X'10'	X'08'	Input	Structure quiesce complete time in STCK format
SS7DSPB	X'18'	X'08'	Input	Start data space/data set capture time in STCK format
SS7DSPE	X'20'	X'08'	Input	End data space capture time in STCK format
SS7RSMB	X'28'	X'08'	Input	Structure resume start time in STCK format
SS7DSE	X'30'	X'08'	Input	End data set capture time in STCK format
SS7CHKE	X'38'	X'08'	Input	Time when all system checkpoints completed in STCK format
SS7PELA	X'3C'	X'04'	Input	Number of allocated elements on primary structure
SS7PELU	X'40'	X'04'	Input	Number of elements in use on primary structure
SS7OELA	X'44'	X'04'	Input	Number of allocated elements on overflow structure
SS7PLEA	X'4C'	X'04'	Input	Number of allocated list entries on primary structure
SS7PLEU	X'50'	X'04'	Input	Number of list entries in use on primary structure
SS7OLEA	X'54'	X'04'	Input	Number of allocated list entries on overflow structure
SS7OLEU	X'58'	X'04'	Input	Number of list entries in use on overflow structure
SS7WRTS	X'5C'	X'04'	Input	Number of SRDS writes required
SS70ELU	X'48'	X'04'	Input	Number of elements in use on overflow structure

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

Register	Contents
1	Address of BPE user exit parameter list (mapped by the BPEUXPL macro). See <i>IMS Version 9: Base Primitive Environment Guide and Reference</i> 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

Register	Contents
15	Return code
0	Always set this to zero.

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).

CQS Structure Event Exit Routine Parameter List

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

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

Field Name	Offset	Length	Field Usage	Description
STXPVSN	X'00'	X'04'	Input	Parameter List Version Number (00000001).
STXEVTN	X'04'	X'04'	Input	Function code 1 Connect Event (STXCONDS).
STXSCODE	X'08'	X'04'	Input	Event Subcode 1 Structure connect (STXCONN). 2 Structure disconnect (STXDISC).
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.
STXSTRVN	X'28'	X'08'	Input	Structure version number (mapped by the CQSSTREX macro).
STXDSTT1	X'34'	X'04'	Input	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 55 for a description of the structure process statistics. For structure disconnect only.

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

Field Name	Offset	Length	Field Usage	Description
STXDSTT2	X'38'	X'04'	Input	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 55 for a description of the CQS request statistics record. For structure disconnect only.
STXDSTT3	X'3C'	X'04'	Input	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 56 for a description of the data object statistics record. For structure disconnect only.
STXDSTT4	X'40'	X'04'	Input	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 57 for a description of the queue name statistics record. For structure disconnect only.
STXDSTT5	X'44'	X'04'	Input	Address of z/OS 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 57 for a description of the z/OS request statistics record. For structure disconnect only.
STXDSTT6	X'48'	X'04'	Input	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 58 for a description of the rebuild statistics record. For structure disconnect only.
STXDSTT7	X'4C'	X'04'	Input	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 60 for a description of the structure checkpoint statistics record. For structure disconnect only.

CQS Structure Event Exit Routine 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

Field Name	Offset	Length	Field Usage	Description
STXPVSN	X'00'	X'04'	Input	Parameter List Version Number (00000001).
STXEVENT	X'04'	X'04'	Input	Structure Event Code 2 Checkpoint Event (STXCHKPT).
STXSCODE	X'08'	X'04'	Input	Structure Event Subcode 1 Structure checkpoint begin (STXCSTRB). 2 Structure checkpoint end (STXCSTRE). 3 Structure checkpoint failure (STXCSTRF). 4 System checkpoint begin (STXCSYSB). 5 System checkpoint end (STXCSYSE). 6 System checkpoint failure (STXCSYSF).
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.
STXCMCQS	X'28'	X'08'	Input	CQS identifier of the master CQS performing the checkpoint process. For system checkpoint, this is the same as the CQS identifier.

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

Field Name	Offset	Length	Field Usage	Description
STXCFLG1	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 (STXC1MST).
N/A	X'31'	X'03'	Input	Reserved.
STXCSTT1	X'34'	X'04'	Input	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 55 for a description of the process statistics record. For system checkpoint end and structure checkpoint end only.
STXCSTT2	X'38'	X'04'	Input	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 55 for a description of the CQS request statistics record. For system checkpoint end and structure checkpoint end only.
STXCSTT3	X'3C'	X'04'	Input	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 56 for a description of the data object statistics record. For system checkpoint end and structure checkpoint end only.
STXCSTT4	X'40'	X'04'	Input	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 57 for a description of the queue name statistics record. For system checkpoint end and structure checkpoint end only.
STXCSTT5	X'44'	X'04'	Input	Address of z/OS 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 57 for a description of the z/OS request statistics record. For system checkpoint end and structure checkpoint end only.
STXCSTT6	X'48'	X'04'	Input	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 58 for a description of the rebuild statistics record. For system checkpoint end and structure checkpoint end only.
STXCSTT7	X'4C'	X'04'	Input	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 60 for a description of the structure checkpoint statistics record. For system checkpoint end and structure checkpoint end only.

CQS Structure Event Exit Routine 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

Field Name	Offset	Length	Field Usage	Description
STXPVSN	X'00'	X'04'	Input	Parameter List Version Number (00000001).
STXEVENT	X'04'	X'04'	Input	Structure Event Code 3 Structure Rebuild Event (STXRBLD).

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

Field Name	Offset	Length	Field Usage	Description	
STXSCODE	X'08'	X'04'	Input	Structure EventSubcode	
				1	Structure rebuild begin (STXRBLB).
				2	Structure rebuild (copy) end (STXCPYE).
				3	Structure rebuild (copy) failure (STXCPYF).
				4	Structure rebuild failure (STXRBLF).
				5	Structure rebuild (recovery) end (STXRCOVE).
6	Structure rebuild (recovery) failure (STXRCOVF).				
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'	This CQS is the master of the process. The CQS identifier and master CQS identifier are the same (STXR1MST).
N/A	X'31'	X'03'	Input	Reserved.	

CQS Structure Event Exit Routine 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

Field Name	Offset	Length	Field Usage	Description	
STXPVSN	X'00'	X'04'	Input	Parameter List Version Number (00000001).	
STXEVENT	X'04'	X'04'	Input	Structure Event Code	
				4	Structure Overflow Event (STXOVFLW).
STXSCODE	X'08'	X'04'	Input	Structure Event Subcode.	
				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).
				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).
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 (STX01MST).
				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)

Field Name	Offset	Length	Field Usage	Description
N/A	X'31'	X'03'	Input	Reserved.
STXOLSTN	X'34'	X'04'	Input	Number of Queue Names entries in the list.
STXOLSTE	X'38'	X'04'	Input	Length of each Queue Name list entry.
STXOLSTA	X'3C'	X'04'	Input	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.

CQS Structure Event Exit Routine 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

Field Name	Offset	Length	Field Usage	Description
STXPVSN	X'00'	X'04'	Input	Parameter List Version Number (00000003).
STXEVENT	X'04'	X'04'	Input	Structure Event Code 5 Structure Status Change Event (STXSCHNG).
STXSCODE	X'08'	X'04'	Input	Structure Event Subcode 1 Structure available again after a loss (STXAVAIL). 2 The structure failed (STXFAIL). 3 CQS lost its connection to the structure (STXLCONN). 4 The log stream is becoming available, making the structure available (STXAVLOG). Important: This subcode applies only to queue structures. 5 The log stream is becoming available, making the structure available (STXFLOG). Important: This subcode applies only to queue structures. 6 The structure failed. It needs to be repopulated because this structure does not support structure recovery (STXREPOP). Important: This subcode applies only to resource structures.
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.
STXSTYPE	X'28'	X'01'	Input	Input structure type (1 queue structure, 2 resource structures).
STXRSTVN	X'40'	X'08'	Input	Input structure version.

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.

See the *IMS Version 9: 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 22. CQS Statistics Header Data

Offset	Length	Field Usage	Description
X'00'	X'08'	Input	Eyecatcher "CQSSTTX"
X'08'	X'04'	Input	Length of header
X'0C'	X'04'	Input	Header version number (00000001)
X'10'	X'04'	Input	Number of structures for which statistics are available
X'14'	X'04'	Input	Number of statistics areas available for each structure
X'18'	X'04'	Input	Length of all statistics areas for each structure
X'1C'	X'04'	Input	Offset to statistics area for first structure (offset from CQSSSTTX)
X'20'	X'04'	Input	CQSSSTAT offset within the statistics area for each structure
X'24'	X'04'	Input	CQSSSTT1 offset within the statistics area for each structure
X'28'	X'04'	Input	CQSSSTT2 offset within the statistics area for each structure
X'2C'	X'04'	Input	CQSSSTT3 offset within the statistics area for each structure
X'30'	X'04'	Input	CQSSSTT4 offset within the statistics area for each structure
X'34'	X'04'	Input	CQSSSTT5 offset within the statistics area for each structure
X'38'	X'04'	Input	CQSSSTT6 offset within the statistics area for each structure
X'3C'	X'04'	Input	CQSSSTT7 offset within the statistics area for each structure
X'40'	X'04'	Input	Reserved
X'44'	X'04'	Input	Reserved
X'48'	X'04'	Input	Reserved
X'4C'	X'04'	Input	Reserved

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 79 for a complete description of all the CQS requests.

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

In this section:

- “Introducing CQS Client Requests”
- “Sequence of CQS Requests Issued by a Client for Queue Structure” on page 70
- “Coding CQS Requests” on page 70
- “CQS Clients and Handling Special Events” on page 77

This section 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:

CQSBWSE	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
CQSINFRM	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
CQSREG	Registers a client with a CQS, establishing communication
CQSRSYNC	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

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 79.

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

Order	Request	Use this request ...
1	CQSREG	To establish communications with CQS.
2	CQSCONN	To connect to a particular structure.
3	CQSRSYNC	To resolve indoubt work with CQS.
4	CQSRECVR ¹	After a CQS cold start to recover specific data objects.
5	CQSINFRM	To register interest in specific queue names.
6	Other CQS requests	To process work. Examples of these other requests are: CQSBWSE, CQSPUT, and CQSREAD.
7	CQSDISC	To disconnect from a structure.
8	CQSSHUT	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.
9	CQSDEREG	To end communications with CQS.

Note:

¹ 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 79) 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 in this section, 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 24. Environment for CQS Requests (Excluding CQSREG and CQSDEREG) Using the Authorized Interface

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

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

Table 25. Environment for CQS Requests (Excluding CQSREG and CQSDEREG) Using the Non-Authorized Interface

Environment Aspect	State
Authorization	Problem state or PSW key 8 (PSW key must match the PSW key when the CQSREG request was issued)
Dispatchable unit mode	Task
Cross memory mode	None (PASN=SASN=HASN)
AMODE	31
ASC Mode	Primary
Home address space	Address space in which CQSREG was issued
Locks	No locks held
Interrupt status	Enabled for interrupts
Control parameters	In primary address space

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

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

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

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

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 75). 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 *cqstoken* on a CQSREAD request, CQS expects the address of the 16-byte CQS token, but for the *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.

```

LA    5,TOKEN
CQSREAD FUNC=READ,CQSTOKEN=(5),...

:
TOKEN DS    XL16

```

Figure 17. Passing an Address for Register

```

L     4,MYBUFLEN
CQSREAD FUNC=READ,BUFSIZE=(4),...

:
MYBUFLEN DC   F'00000024'

```

Figure 18. Passing a value 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.

```

CQSREAD FUNC=READ,CQSTOKEN=TOKENADR,...

:
TOKENADR DC   A(TOKEN) TOKEN    DS    XL16

```

Figure 19. Passing an Address for Symbol

```

CQSREAD FUNC=READ,BUFSIZE=MYBUFLEN,...

:
MYBUFLEN DC   F'00000024'

```

Figure 20. Passing a Value for Symbol

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.

```

CQSREAD FUNC=READ,CQSTOKEN=@(TOKEN),...
:
TOKEN DC XL16'0000A765B55CFF00'

```

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 4 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 EQU$

```

Figure 23. Coding CQSREAD with the OPTWORD1 parameter

Using an ECB with CQS Requests

Some requests allow you to use a z/OS 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.

Related Reading: For information on using an ECB, see the *z/OS MVS Programming: Authorized Assembler Services Guide*.

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 this. 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 76. 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 28. Return and Reason Codes for Errors Detected by the CQS Interface

Return Code	Reason Code	Meaning
X'00000008'	X'00000210'	The <i>cqstoken</i> is invalid.
X'00000008'	X'00000214'	The <i>connecttoken</i> is invalid.
X'00000010'	X'00000430'	The CQS address space is not available.
X'00000014'	X'00000600'	The CQS interface is unable to access internal blocks.
X'00000014'	X'00000604'	The client is running in problem state or is using an incorrect PSW key.
X'00000014'	X'00000608'	The client passed an invalid function code to the CQS interface.
X'00000014'	X'0000060C'	The client specified an invalid CQS request type.
X'00000014'	X'00000610'	CQS was unable to allocate storage to copy the request parameters.
X'00000014'	X'00000614'	The total length of all request parameters passed was less than the sum of all parameter lengths.
X'00000014'	X'00000618'	The value passed to the interface for the total length of all parameters was either zero or negative.
X'00000014'	X'0000061C'	The value passed to the interface for the total parameter count was either zero or negative.
X'00000014'	X'00000620'	The length of one of the request's parameters was negative.
X'00000014'	X'00000624'	The length passed for the structure-call parameter list was invalid.
X'00000014'	X'00000628'	Invalid request function code.
X'00000014'	X'0000062C'	Invalid request parameter list version number.
X'00000014'	X'00000630'	An incorrect number of parameters was passed for the requested function.
X'00000014'	X'00000634'	A parameter was passed with an incorrect length.
X'00000014'	X'00000638'	A parameter was passed by value instead of by address.
X'00000014'	X'0000063C'	A parameter was passed by address instead of by value.
X'00000014'	X'00000640'	The CQS request abended before being sent to the CQS.
X'00000014'	X'00000644'	The CQS request abended while CQS was copying the request parameters. This error is usually caused by the client's passing bad parameter data.
X'00000014'	X'00000648'	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.

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*

Clients assembled using IMS Version 6 request macros can register with either an IMS Version 6 or IMS Version 7 CQS.

Attention: Clients assembled using IMS Version 7 macros can only register with an IMS Version 7 CQS.

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 `CQSRSYNC` request 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 non-empty. 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 CQSBWSE 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 34 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 35 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 101.
- For more information on the CQSSHUT request, see “CQSSHUT Request” on page 149.

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 121, see “CQSDEL Request” on page 96, and see “CQSINFRM Request” on page 106.

Chapter 6. CQS Client Requests

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

- “CQSBRWSE Request” on page 80
- “CQSCHKPT Request” on page 87
- “CQSCONN Request” on page 90
- “CQSDEL Request” on page 96
- “CQSDEREG Request” on page 100
- “CQSDISC Request” on page 101
- “CQSINFRM Request” on page 106
- “CQSMOVE Request” on page 110
- “CQSPUT Request” on page 114
- “CQSQUERY Request” on page 121
- “CQSREAD Request” on page 130
- “CQSRECVR Request” on page 135
- “CQSREG Request” on page 140
- “CQSRSYNC Request” on page 142
- “CQSSHUT Request” on page 149
- “CQSUNLCK Request” on page 150
- “CQSUPD Request” on page 155

This section 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.

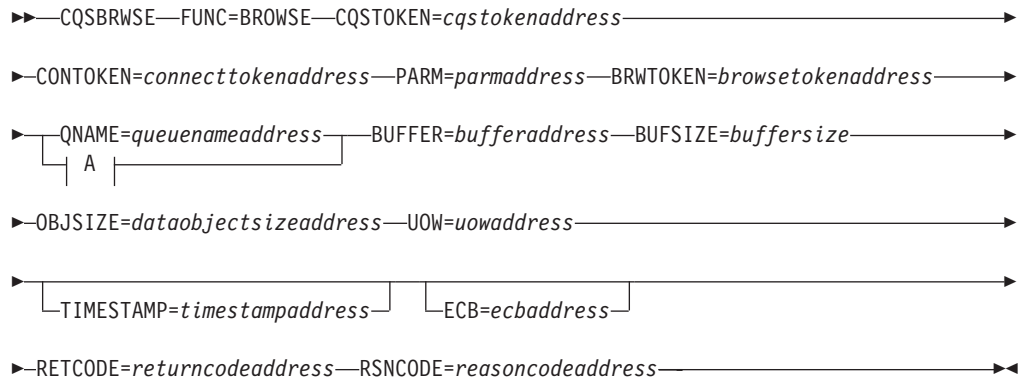
Use these requests if you are writing or maintaining a CQS client. You do not need to use them if you are using an IBM-supplied client, such as an IMS control region.

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.

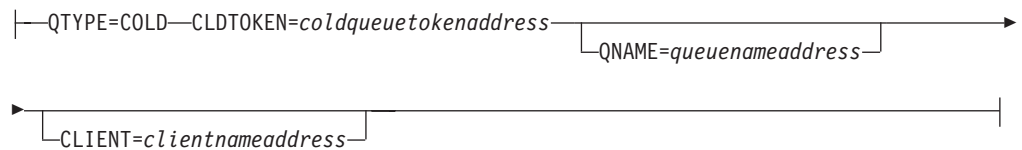
CQSBWSE Request

Format for CQSBWSE

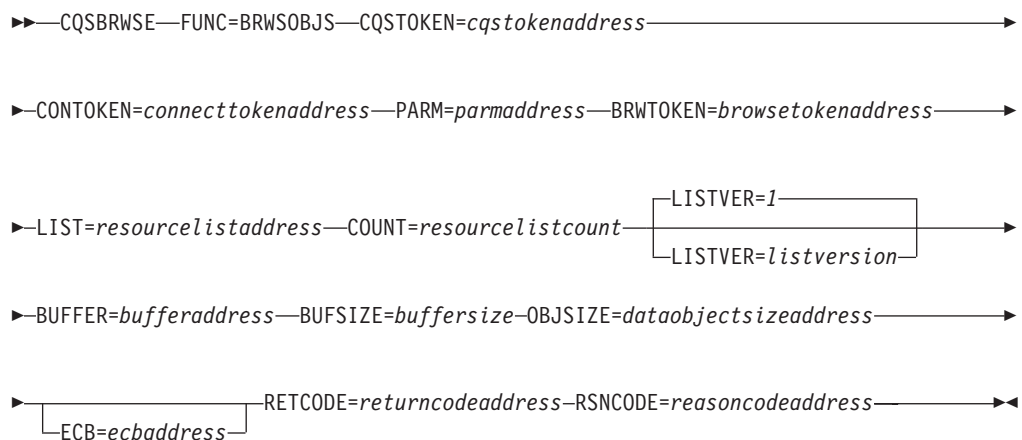
BROWSE Function of CQSBWSE: Use the BROWSE function of a CQSBWSE request to retrieve a copy of a data object from a specific queue.



A:



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



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

```

▶▶—CQSBWSE—FUNC=COMPLETE—CQSTOKEN=cqstokenaddress—————▶▶
▶—CONTOKEN=connecttokenaddress—PARM=parmaddress—BRWTKEN=browsetokenaddress————▶
▶—[ECB=ecbaddress]—RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress————▶▶

```

CONTINUE Function of CQSBWSE: Use the CONTINUE function of a CQSBWSE request if a previous CQSBWSE request retrieved partial data and you want to retrieve the rest of the data object.

```

▶▶—CQSBWSE—FUNC=CONTINUE—CQSTOKEN=cqstokenaddress—————▶▶
▶—CONTOKEN=connecttokenaddress—PARM=parmaddress—BRWTKEN=browsetokenaddress————▶
▶—BUFFER=bufferaddress—BUFSIZE=buffersize—OBJSIZE=dataobjectsizedaddress————▶▶
▶—[ECB=ecbaddress]—RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress————▶▶

```

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

```

▶▶—CQSBWSE—FUNC=DSECT—————▶▶

```

Usage of CQSBWSE

A CQSBWSE FUNC=BROWSE request retrieves a copy of a data object from a specific queue on a queue structure. The first CQSBWSE 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 CQSBWSE FUNC=BROWSE request retrieves a copy of the next data object. The data object is returned in the client buffer provided on the CQSBWSE 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 CQSBWSE FUNC=BROWSE request with a zero browse token passes back the first data object. A CQSBWSE 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 CQSBWSE FUNC=BRWSOBS request retrieves information on one or more data objects from a resource structure. The first CQSBWSE FUNC=BRWSOBS 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 CQSBWSE request. Each subsequent CQSBWSE FUNC=BRWSOBS request retrieves the next set of data object entries. A browse token maintains the cursor position of the data objects being browsed. A CQSBWSE FUNC=BRWSOBS request with a zero browse token retrieves information on as many data objects as fit in the buffer. A CQSBWSE FUNC=BRWSOBS 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=BRWSOBS request. The CQSBRWSE FUNC=CONTINUE request is not supported for a resource structure because CQSBRWSE FUNC=BRWSOBS 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, CQSDDEL, 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:

BRWTKEN=*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=BRWSOBS request should be passed as input on a subsequent CQSBRWSE=BROWSE, CONTINUE, COMPLETE, or BRWSOBS 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, BRWTKEN 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=BRWSOBS, 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 resourceid, the completion code, resourceid 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 resourceid status indicates how the resourceid 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 resourceid. This data object entry contains the resourceid 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 resourceid. 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 resourceid 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 is invalid. The name type must be a decimal number from 1 to 255.

X'00000024'

CQS internal error.

X'00000040'

No resources matching either resourceid, resource type, owner, or some combination of these, were 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.

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 8-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=*resourcelistcount*

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 z/OS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise it is processed synchronously.

LIST=*resourcelistaddress*

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 CQSBRWSL 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 resourceid can be a wildcard parameter. The resourceid is unique in the IMSplex. The resourceid 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 resourceid, you should also specify the resource type to enhance performance. You must specify the resourceid, 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, resourceid, 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 resourceid 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=*dataobjectsizeaddress*

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=BRWSOBS 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=BRWSOBS 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.

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 CQSBRWSE_PARM_LEN (defined using the FUNC=DSECT request).

QNAME=*queuenameaddress*

Four-byte output parameter that specifies the address of a 16-byte queue name field.

For a CQSBWSE 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 CQSBWSE requests, the queue name field is an input field that specifies the queue name from which the data object is retrieved for all CQSBWSE 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 CQSBWSE 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 CQSBWSE reason code.

TIMESTAMP=*timestampaddress*

Four-byte output parameter that specifies the address of an 8-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 CQSBWSE

Table 29 lists the return and reason code combinations that can be returned for CQSBWSE requests. Use a CQSBWSE FUNC=DSECT request to include equate statements in your program for the return and reason codes.

Table 29. CQSBWSE Return and Reason Codes

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000004'	X'00000120'	The buffer size (<i>buffersize</i>) is less than the data object size (<i>dataobjectsize</i>). Partial data is returned.
X'00000004'	X'00000124'	The buffer size (<i>buffersize</i>) is too small to contain the next resource data object entry. No partial data is returned.
X'00000004'	X'00000128'	No data object to retrieve on queue name (<i>queuename</i>) specified.
X'00000004'	X'0000012C'	No partial data to return.
X'00000004'	X'00000138'	Request complete and the last data object is returned.

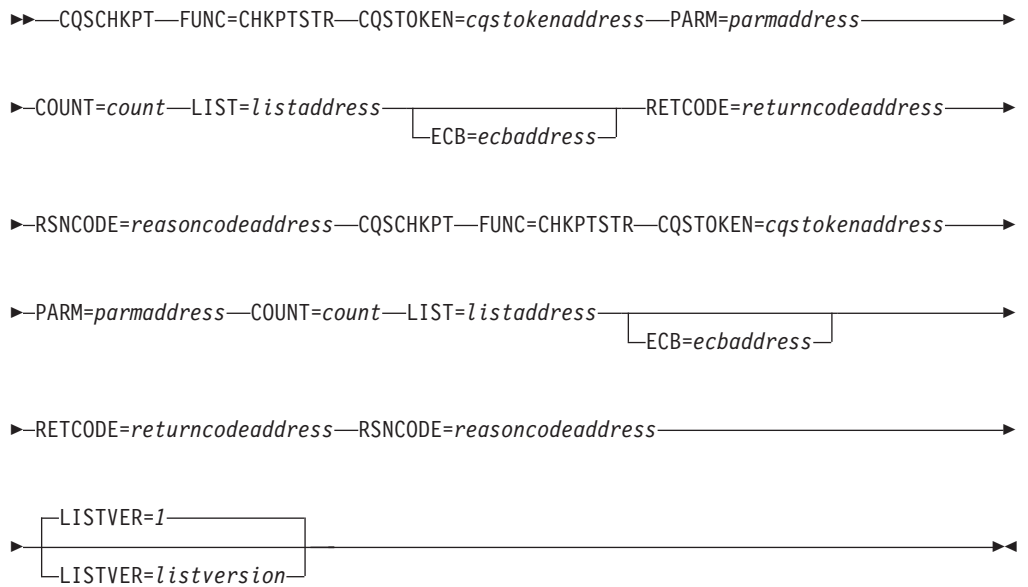
Table 29. CQSBRWSE Return and Reason Codes (continued)

Return Code	Reason Code	Meaning
X'00000004'	X'0000013C'	No more data objects to return.
X'00000008'	X'00000210'	<i>cqstoken</i> is invalid.
X'00000008'	X'00000214'	<i>connecttoken</i> is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'0000021C'	<i>browsetoken</i> is invalid.
X'00000008'	X'00000220'	<i>queueName</i> is invalid.
X'00000008'	X'00000224'	<i>buffer</i> is invalid.
X'00000008'	X'00000228'	<i>bufferSize</i> is invalid.
X'00000008'	X'0000022C'	<i>dataObjectsSize</i> is invalid.
X'00000008'	X'00000230'	<i>uow</i> is invalid.
X'00000008'	X'00000234'	<i>browsetoken</i> is invalid.
X'00000008'	X'00000250'	Count is invalid.
X'00000008'	X'00000254'	List address is invalid.
X'00000008'	X'0000027C'	CQSBRWSE FUNC=BROWSE is not allowed for a resource structure. CQSBRWSE FUNC=CONTINUE is not allowed for a resource structure. No partial data is returned from a resource structure.
X'00000008'	X'00000280'	CQSBRWSE FUNC=BRWSOBS is not allowed for a queue structure.
X'00000008'	X'00000284'	Parmlist version is invalid.
X'00000008'	X'00000288'	List version is invalid.
X'00000010'	X'00000400'	A CQSRSYNC is required for this structure.
X'00000010'	X'00000404'	Structure is inaccessible. Retry request later.
X'00000010'	X'00000408'	Current position lost, reissue CQSBRWSE request.
X'00000010'	X'00000430'	No CQS address space.
X'00000014'	X'00000500'	CQS internal error.

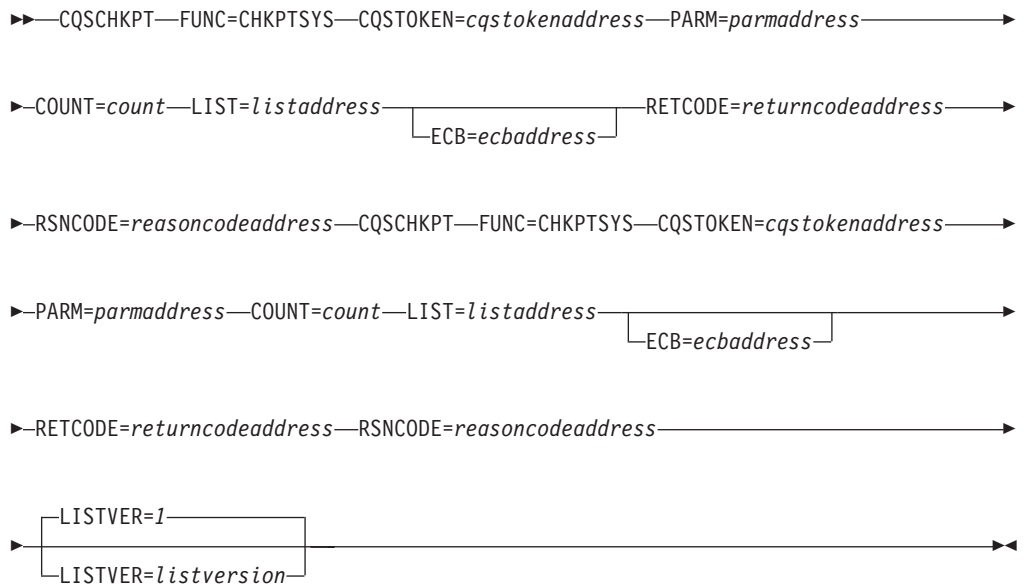
CQSCHKPT Request

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.



CHKPTSYS Function of CQSCHKPT: Use the CHKPTSYS function of a CQSCHKPT request to initiate a CQS system checkpoint.



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.



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 z/OS 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.

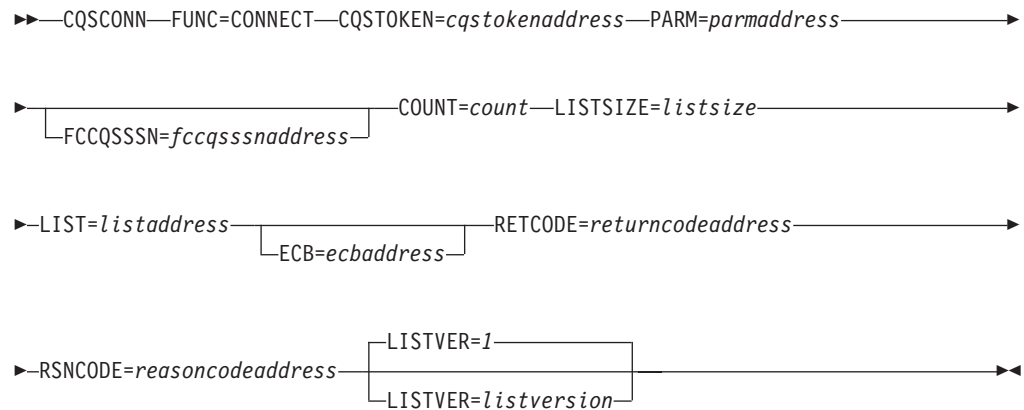
Table 30. CQSCHKPT Return and Reason Codes

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000008'	X'00000210'	<i>cqstoken</i> is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000250'	<i>count</i> is invalid.
X'00000008'	X'00000254'	<i>listaddress</i> is invalid.
X'00000008'	X'00000284'	Parmlist version is invalid.
X'00000008'	X'00000288'	List version is invalid.
X'0000000C'	X'00000300'	Request succeeded for at least one, but not all, list entries. See <i>compcode</i> for individual errors.
X'0000000C'	X'00000304'	Request failed for all list entries. See <i>compcode</i> for individual errors.
X'00000010'	X'0000040C'	CQS shutdown is pending. Client-initiated checkpoint requests are not allowed.
X'00000010'	X'00000430'	No CQS address space.

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 FCCQSSN 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 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 z/OS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise it is processed synchronously.

FCCQSSSN=*fccqsssnaddress*

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' | <i>structurename</i> 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. |

X'00000020'	structureinformexit was specified but is not allowed for a resource structure.
X'00000024'	structureinformparm was specified but is not allowed for a resource structure.
X'0000002C'	structureattributes was specified but is not allowed for a resource structure.
X'00000030'	Qtype was specified but is not allowed for a resource structure.
X'00000034'	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 <i>structureattributes</i> 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 3 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=*listsize*

Four-byte input parameter that specifies the size of the connect list. *listsize* specifies the total length of all entries in the list, not the length of a single entry.

LISTVER=1 | listversion

Input parameter that specifies an equate for the list version. Use the DSECT function of a CQSCONN request to include equate (EQU) statements in your program for the CQSCONN 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 CQSCONN_PARM_LEN (defined using the FUNC=DSECT request).

RETCODE=*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=*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 31 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.

Table 31. CQSCONN Return and Reason Codes

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000004'	X'00000100'	The client was previously connected to one or more of the specified structures through this CQS. Client is connected to all structures.
X'00000008'	X'00000210'	<i>cqstoken</i> is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000250'	<i>count</i> is invalid.
X'00000008'	X'00000254'	<i>listaddress</i> is invalid.
X'00000008'	X'00000258'	<i>listsize</i> is invalid.
X'00000008'	X'00000284'	Parmlist version is invalid.
X'00000008'	X'00000288'	List version is invalid.
X'0000000C'	X'00000300'	Request succeeded for one but not all list entries. See <i>compcode</i> for individual errors.
X'0000000C'	X'00000304'	Request failed for all list entries. See <i>compcode</i> for individual errors.

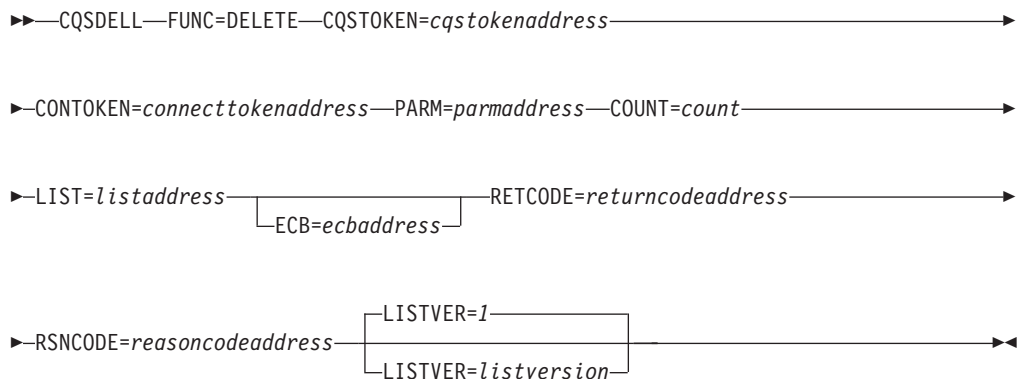
Table 31. CQSCONN Return and Reason Codes (continued)

Return Code	Reason Code	Meaning
X'00000010'	X'0000040C'	CQS shutdown in progress (CQSSHUT). CQS is waiting for all clients to disconnect, and no new client connections are allowed.
X'00000010'	X'00000410'	The maximum number of clients are connected to this CQS. This request would exceed the client connection limit. No further client connections are allowed.
X'00000010'	X'00000430'	No CQS address space.
X'00000014'	X'00000500'	CQS internal error.

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.



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.



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 resourceid, 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.

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 z/OS 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 resourceid 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 <i>deleteqpos</i> (Delete type 2).
X'00000008'	Invalid <i>deletetype</i> .
X'0000000C'	Invalid <i>locktoken</i> (Delete type 1).
X'00000010'	Invalid <i>queuename</i> (Delete type 2 or type 3).
X'00000014'	Invalid <i>uow</i> (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 <i>queuename</i> 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'	<i>Resourceid</i> is invalid. The name type must be a decimal number from 1 to 255.
X'00000034'	<i>Version</i> does not match that of an existing resource.
X'00000038'	<i>Resourcetype</i> is invalid. The resource type must be a decimal number from 1 to 255.
X'0000003C'	<i>Owner</i> is invalid. The owner is required for delete type 5.
X'00000040'	<i>Version</i> 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 *resourceytp* 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 100 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.

Table 32. CQSDEL Return and Reason Codes

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000008'	X'00000210'	<i>cqstoken</i> is invalid.
X'00000008'	X'00000214'	<i>connecttoken</i> is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000250'	<i>count</i> is invalid.
X'00000008'	X'00000254'	<i>listaddress</i> is invalid.
X'00000008'	X'00000284'	Parmlist version is invalid.
X'00000008'	X'00000288'	List version is invalid.
X'0000000C'	X'00000300'	Request succeeded for at least one, but not all, list entries. See <i>compcode</i> for individual errors.
X'0000000C'	X'00000304'	Request failed for all list entries. See <i>compcode</i> for individual errors.
X'00000010'	X'00000400'	A CQSRSYNC is required for this structure.
X'00000010'	X'00000430'	No CQS address space.
X'00000014'	X'00000500'	CQS internal error.

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*—PARAM=*parmaddress*—►►

►►—RETCODE=*returncodeaddress*—RSNCODE=*reasoncodeaddress*—►►

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

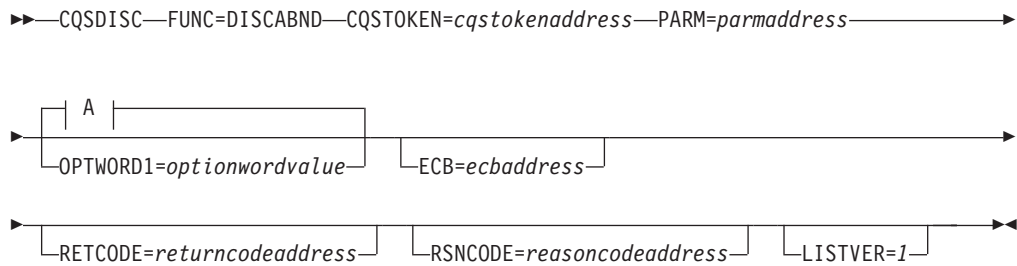
Table 33 lists the return and reason code combinations that can be returned for CQSDEREG requests.

Table 33. CQSDEREG Return and Reason Codes

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000004'	X'00000104'	Unable to free CQS's storage in client's address space. The <i>cqstoken</i> is now invalid.
X'00000004'	X'00000108'	Unable to delete z/OS Resource Manager routine. The <i>cqstoken</i> is now invalid.
X'00000008'	X'00000210'	<i>cqstoken</i> is invalid.
X'00000008'	X'00000248'	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.
X'00000008'	X'00000284'	Parmlist version is invalid.
X'00000010'	X'00000434'	Request is active.
X'00000014'	X'00000500'	CQS internal error. The <i>cqstoken</i> is now invalid.
X'00000014'	X'00000504'	Storage allocation error for work area.
X'00000014'	X'00000518'	CQS internal error (unable to create ESTAE).

CQSDISC Request**Format for CQSDISC**

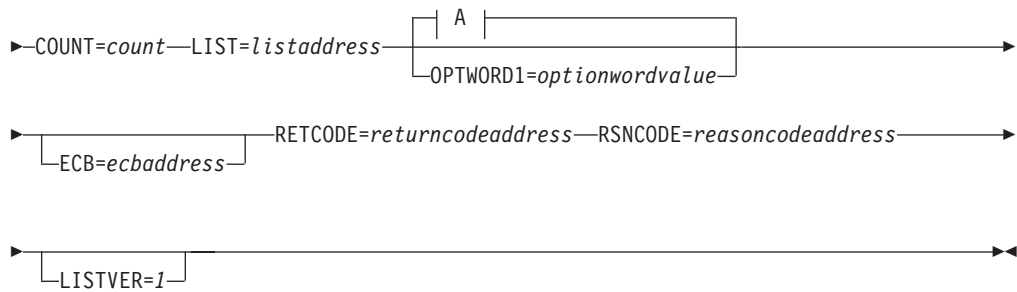
DISCABND Function of CQSDISC: Use the DISCABND function of a CQSDISC request while the client is terminating abnormally to terminate client connections to all coupling facility structures.



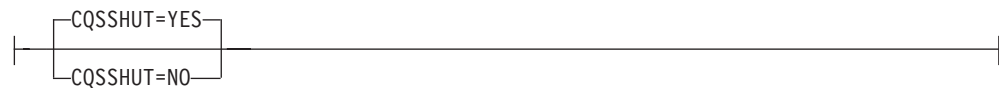
A:



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.



A:



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  CQSSHUT=YES
CQSDISC_SHUTNEQX  CQSSHUT=NO
```

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 z/OS 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 3 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

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000004'	X'00000130'	Request completed successfully for the requested structures. Client is still connected to additional coupling facility structures.
X'00000004'	X'00000140'	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'.

Table 34. CQSDISC Return and Reason Codes (continued)

Return Code	Reason Code	Meaning
X'00000008'	X'00000210'	<i>cqstoken</i> is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000250'	<i>count</i> is invalid.
X'00000008'	X'00000254'	<i>listaddress</i> is invalid.
X'00000008'	X'00000284'	Parmlist version is invalid.
X'00000008'	X'00000288'	List version is invalid.
X'0000000C'	X'00000300'	Request succeeded for at least one but not all list entries. See <i>compcode</i> for individual errors.
X'0000000C'	X'00000304'	Request failed for all list entries. See <i>compcode</i> for individual errors.
X'00000010'	X'00000430'	No CQS address space.

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.

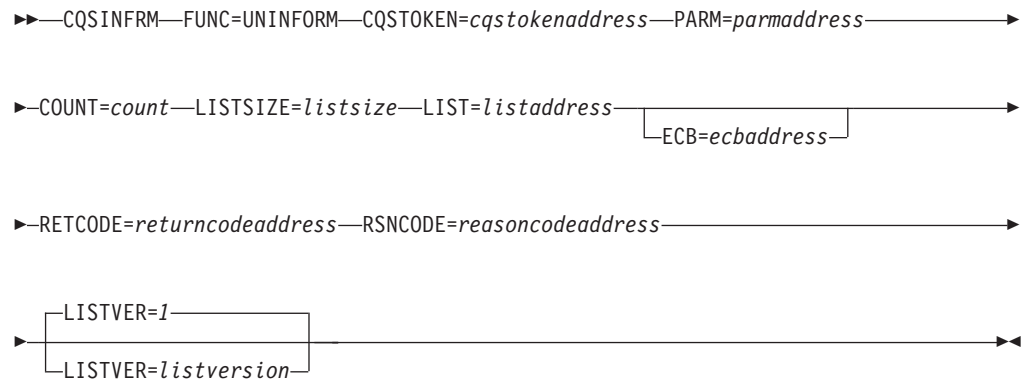
▶▶—CQSINFRM—FUNC=INFORM—CQSTOKEN=*cqstokenaddress*—PARAM=*parmaddress*—▶▶

▶—COUNT=*count*—LISTSIZE=*listsize*—LIST=*listaddress*—▶
└—ECB=*ecbaddress*—┘

▶—RETCODE=*returncodeaddress*—RSNCODE=*reasoncodeaddress*—▶▶

▶—LISTVER=*l*—▶▶
└—LISTVER=*listversion*—┘

UNIFORM Function of CQSINFRM: Use the UNIFORM 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 175.

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=UNIFORM 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 z/OS 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'	<i>connecttoken</i> is invalid.
X'00000014'	<i>queuelistcount</i> 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 <i>queuelist</i> . See <i>queuecompletioncode</i> for individual errors.
X'00000024'	Request failed for all queues in <i>queuelist</i> . See <i>queuecompletioncode</i> for individual errors or successes.
X'00000030'	A CQSRSYNC 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)s 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.

RSNCODE=*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 35. CQSINFRM Return and Reason Codes

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000004'	X'00000134'	Request completed successfully. One or more queues have work.
X'00000008'	X'00000210'	<i>cqstoken</i> is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000250'	<i>count</i> is invalid.
X'00000008'	X'00000254'	<i>listaddress</i> is invalid.
X'00000008'	X'00000258'	<i>listsize</i> is invalid.
X'00000008'	X'00000284'	Parmlist version is invalid.
X'00000008'	X'00000288'	List version is invalid.
X'0000000C'	X'00000300'	Request succeeded for at least one, but not all, list entries. Check <i>structurecompletioncode</i> for individual errors or successes.
X'0000000C'	X'00000304'	Request failed for all list entries. See <i>structurecompletioncode</i> for individual errors.
X'00000010'	X'00000430'	No CQS address space.

CQSMOVE Request

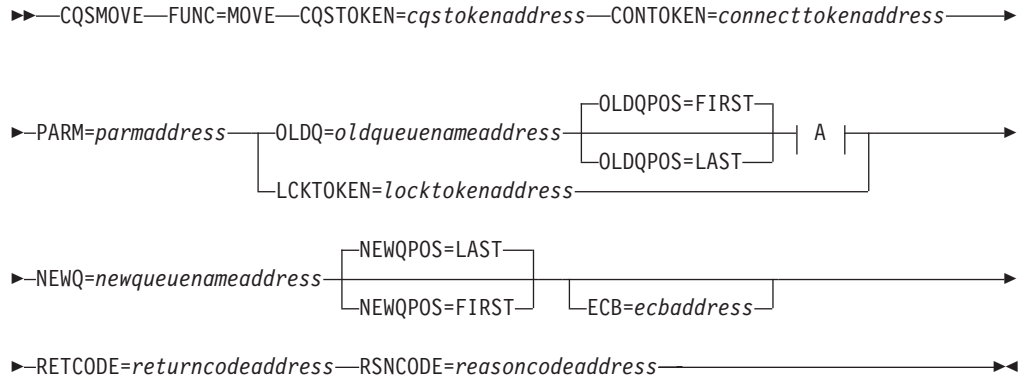
Format for CQSMOVE

DSECT Function of CQSMOVE: Use the DSECT function of a CQSMOVE request to include equate (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 OPTWORD1 parameter.

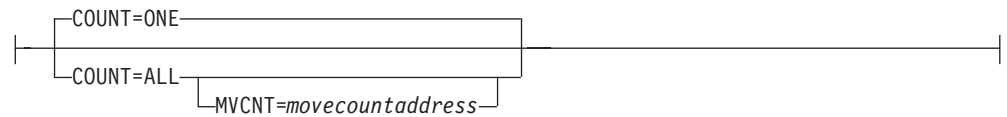
▶▶—CQSMOVE—FUNC=DSECT—◀◀

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.

MOVE Function of CQSMOVE using Literal Parameters

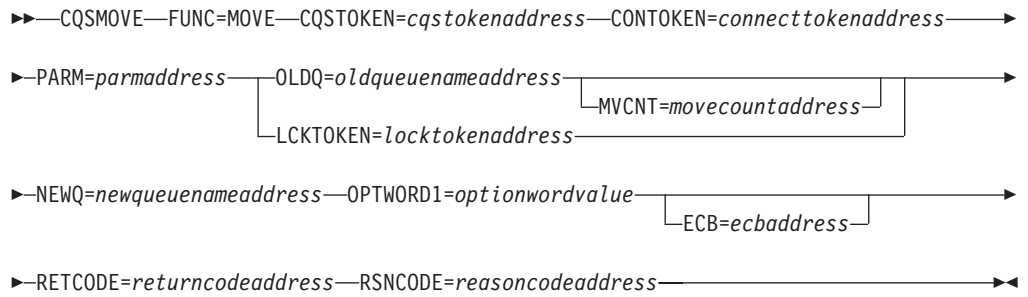


A:



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



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 z/OS 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=newqueueaddress

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=oldqueueaddress

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=optionwordvalue

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

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000004'	X'00000128'	No data object to move for queue name specified.
X'00000008'	X'00000210'	<i>cqstoken</i> is invalid.
X'00000008'	X'00000214'	<i>connecttoken</i> is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'0000021C'	<i>locktoken</i> is invalid.
X'00000008'	X'00000220'	Queue name is invalid.
X'00000008'	X'00000224'	Buffer address is invalid.
X'00000008'	X'0000027C'	CQSMOVE is not allowed for a resource structure.
X'00000008'	X'00000284'	Parmlist version is invalid.

Table 36. CQSMOVE Return and Reason Codes (continued)

Return Code	Reason Code	Meaning
X'00000010'	X'00000400'	A CQSRSYNC is required for this structure.
X'00000010'	X'00000404'	Structure is inaccessible. Retry request later.
X'00000010'	X'00000414'	Unable to move the data object because the destination queue is full. CQSMOVE requests for other queues are allowed.
X'00000010'	X'0000041C'	Request pending. A structure recovery or CQS restart might be required to complete.
X'00000010'	X'00000430'	No CQS address space.
X'00000010'	X'00000440'	Locked (nonrecoverable) data object lost due to rebuild.
X'00000014'	X'00000500'	CQS internal error.

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—▶
▶—PARAM=paramaddress—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—▶▶

```

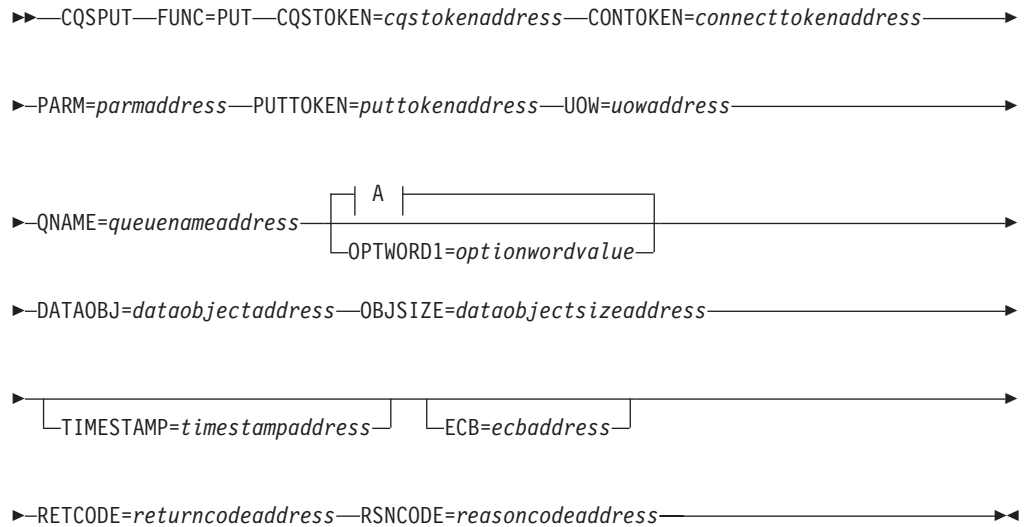
FORGET Function of CQSPUT: Use the FORGET function of a CQSPUT request to discard any information CQS has on a committed unit of work.

```

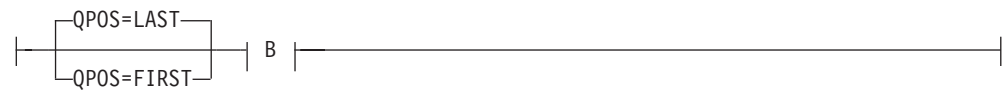
▶▶—CQSPUT—FUNC=FORGET—CQSTOKEN=cqstokenaddress—▶
▶—CONTOKEN=connecttokenaddress—PARAM=paramaddress—PUTTOKEN=puttokenaddress—▶
▶—RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress—▶▶
└─ECB=ecbaddress─┘

```

PUT Function of CQSPUT: Use the PUT function of a CQSPUT request to place a data object on a queue.



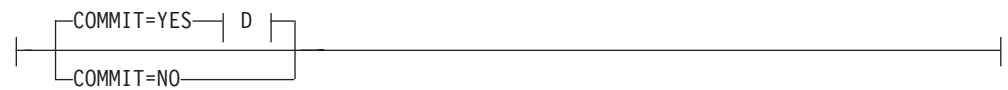
A:



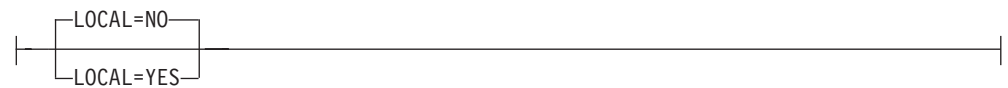
B:



C:



D:



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 data object is put on a queue, a timestamp is stored with the data object. The source of the timestamp is based on whether `TIMESTAMP=` is used on the `CQSPUT=` request. If `TIMESTAMP=` is specified on the `CQSPUT` request, the value specified for `TIMESTAMP=` is stored with the data object. If `TIMESTAMP=` is not specified on the `CQSPUT` request, a timestamp representing the current time is generated and stored with the data object. The timestamp is returned on the `CQSQUERY FUNC=QTYPE` request if it is associated with the oldest data object on the queue or the newest data object on the queue.

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=*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.

DATAOBJ=*dataobjectaddress*

Four-byte input parameter that specifies the address of the client data object to be placed on the specified queue.

ECB=*ecbaddress*

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

LOCAL=NO | N | YES | Y

Input parameter that indicates whether the client should keep a local copy of the data.

NO

Indicates the client wants CQS to place the data object on the specified client queue and make the object available to other CQs.

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:

```
CQSPUT_LCLYEQUX LOCAL=YES
CQSPUT_LCLNEQUX LOCAL=NO
```

OBJSIZE=*dataobjectsizeaddress*

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').

OPTWORD1=*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.

FIRST The data object is added to the beginning of the queue.

LAST The data object is added to the end 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 38. CQSPUT Return and Reason Codes

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000008'	X'00000210'	<i>cqstoken</i> is invalid.
X'00000008'	X'00000214'	<i>connecttoken</i> is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'0000021C'	<i>puttoken</i> is invalid.
X'00000008'	X'00000220'	<i>queuename</i> is invalid.
X'00000008'	X'00000224'	<i>dataobject</i> is invalid.
X'00000008'	X'00000228'	<i>dataobjectsize</i> is invalid.
X'00000008'	X'00000230'	<i>uow</i> is invalid.
X'00000008'	X'00000238'	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.

Table 38. CQSPUT Return and Reason Codes (continued)

Return Code	Reason Code	Meaning
X'00000008'	X'00000260'	A CQSPUT FUNC=PUT request was issued, but the unit of work was already committed.
X'00000008'	X'00000264'	A CQSPUT FUNC=FORGET request was issued for a recoverable unit of work, but the unit of work was not committed.
X'00000008'	X'00000268'	A CQSPUT FUNC=ABORT request was issued for a nonrecoverable unit of work.
X'00000008'	X'0000026C'	A CQSPUT FUNC=ABORT request was issued for a recoverable unit of work but the unit of work was already committed.
X'00000008'	X'00000270'	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.
X'00000008'	X'00000274'	RECOVERABLE=NO was specified for a unit of work that was indicated as recoverable on a previous CQSPUT FUNC=PUT request.
X'00000008'	X'0000027C'	CQSPUT is not allowed for a resource structure.
X'00000008'	X'00000284'	Parmlist version is invalid.
X'00000010'	X'00000400'	A CQSRSYNC is required for this structure.
X'00000010'	X'00000404'	Structure inaccessible. Retry request later.
X'00000010'	X'00000414'	Queue for <i>queuename</i> 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.
X'00000010'	X'00000418'	Structure is full. All CQSPUT requests are rejected.
X'00000010'	X'00000430'	No CQS address space.
X'00000014'	X'00000500'	CQS internal error.

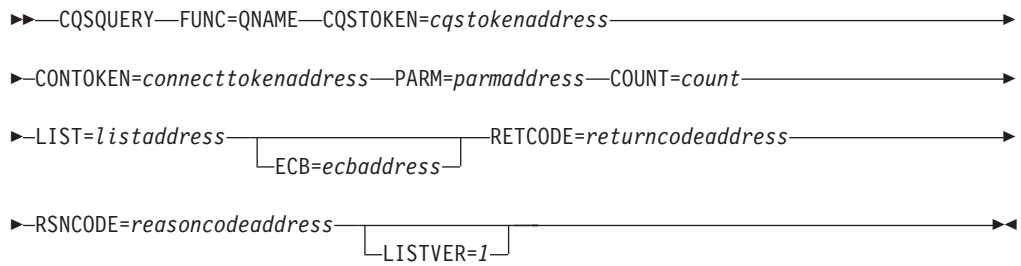
CQSQUERY Request

Format for CQSQUERY

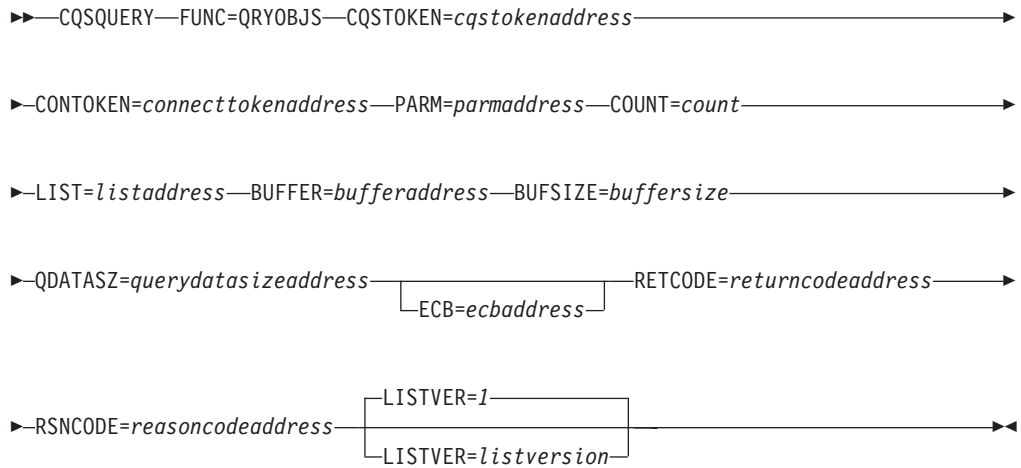
DSECT Function of CQSQUERY: Use the DSECT function of a CQSQUERY request to include equate (EQU) statements in your program for the CQSQUERY parameter list length and CQSQUERY return and reason codes.

►►—CQSQUERY—FUNC=DSECT—◄◄

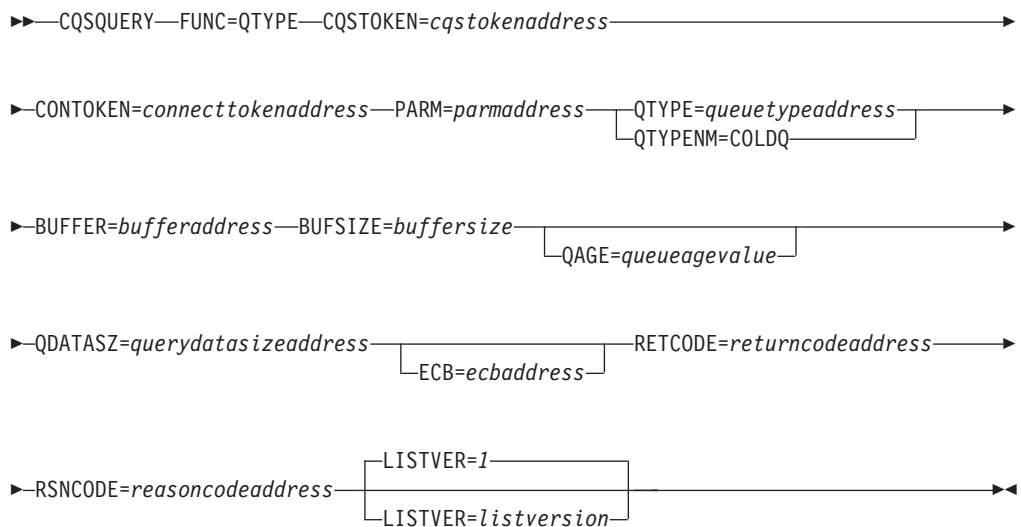
QNAME Function of CQSQUERY: Use the QNAME function of a CQSQUERY request to retrieve information about a specific queue managed by CQS.



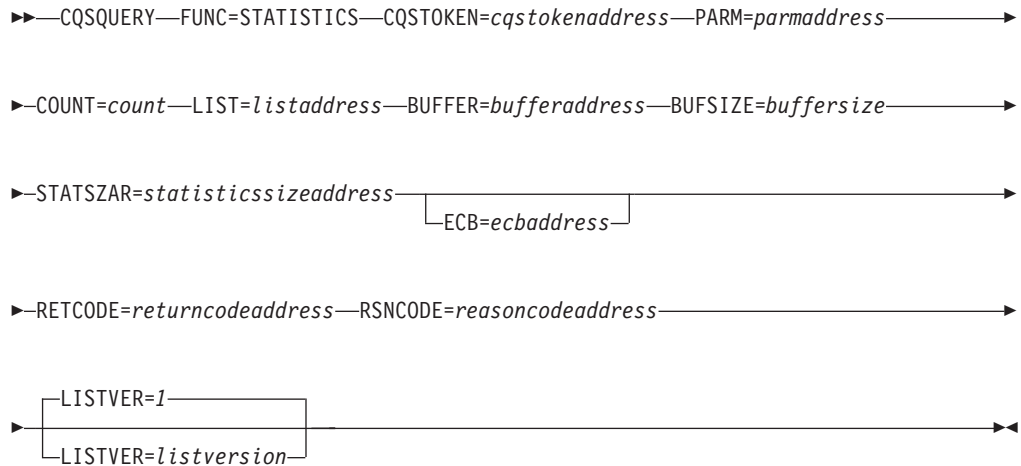
QRYOBS Function of CQSQUERY: Use the QRYOBS function of a CQSQUERY request to retrieve the queue counts for a specified list of queue names.



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.

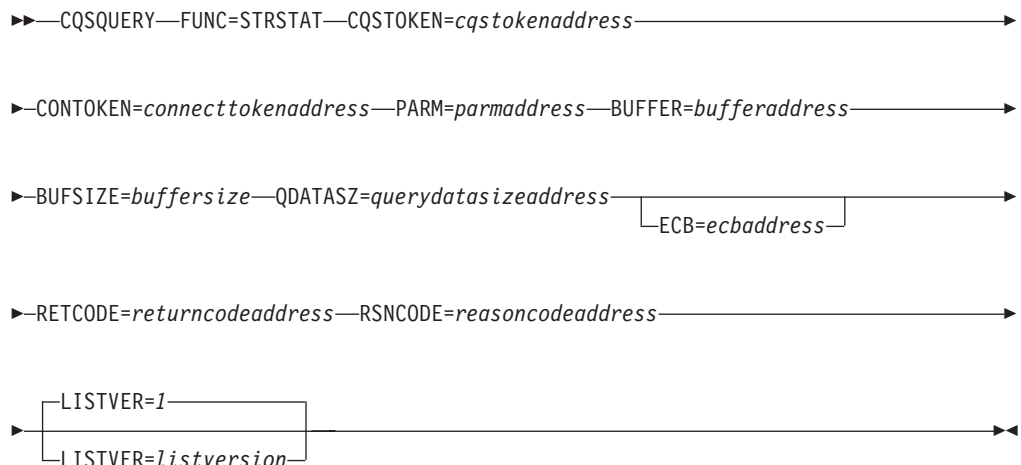


STATISTICS Function of CQSQUERY: Use the STATISTICS function of a CQSQUERY request to retrieve status information on all the queues managed by CQS.



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.



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=QRYOBS 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. A CQSQUERY

FUNC=STRSTAT 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 queue name.

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 queue name. If the completion code is 0, then the queue names that match the input queue name and their queue counts 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 queue count found for the queue name. If the queue name is a wildcard parameter, this queue count is the total queue counts of all the queue names that match the wildcard parameter. An entry for each queue name that is a match is passed in the output buffer along with the queue count for the queue name. 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 queue type, and thus could have a significant performance impact.

CQSQUERY FUNC=QTYPE: For CQSQUERY FUNC=QTYPE, information about all the queues in the queue type is returned, including the queue name, data object count, 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 queue type, 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
- z/OS 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 here 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 CQSQRQT DSECT. For CQSQUERY FUNC=STATISTICS, the buffer is mapped by the CQSQRST DSECT. For CQSQUERY FUNC=STRSTAT, the buffer is mapped by the CQSQSTAT DSECT. For CQSQUERY FUNC=QRYOBS, the buffer is mapped by the CQSQRQO DSECT.

BUFSIZE=*buffer size*

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 z/OS 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 CQSRYL 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:

compcode

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:

compcode

Four-byte output field to receive the completion code from the request. Possible completion codes are:

X'00000000'	Request completed successfully.
X'00000008'	<i>connecttoken</i> 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'0000'	Request completed successfully. A list of resources that match the qname and their queue counts are returned in the output buffer.
X'0004'	qname is invalid.
X'0010'	qname does not have any objects. The queue count is zero.
X'0020'	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'0024'	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.

queuname

Sixteen-byte input parameter that specifies the queue name for which data object count information is to be retrieved. This parameter is required. The queuname 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 queuname 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=querydatasizeaddress

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).

QTYPENM=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.

RSNCODE=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.

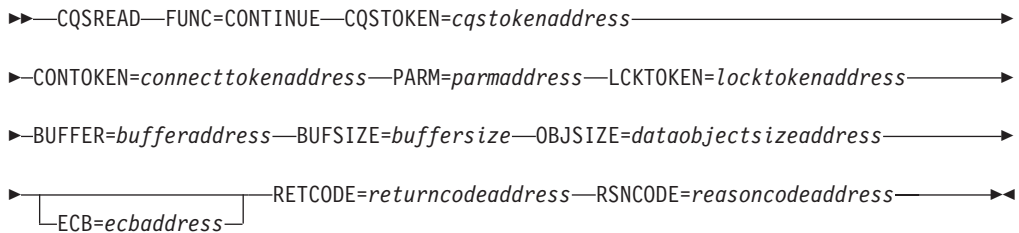
Table 39. CQSQUERY Return and Reason Codes

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000004'	X'00000120'	The buffer size (<i>buffer size</i>) is less than the query-data size (<i>querydatasize</i>). Partial data is returned. <i>querydatasize</i> points to the actual buffer size needed to contain all the data.
X'00000004'	X'00000124'	<i>buffer size</i> is too small to contain data for number of entries specified in <i>list</i> .
X'00000004'	X'00000128'	No data objects on queue type.
X'00000008'	X'00000210'	<i>cqstoken</i> is invalid.
X'00000008'	X'00000214'	<i>connecttoken</i> is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000224'	<i>buffer address</i> is invalid.
X'00000008'	X'00000228'	<i>buffer size</i> is invalid.
X'00000008'	X'0000022C'	<i>statisticssize</i> or <i>querydatasize</i> is invalid.
X'00000008'	X'0000023C'	<i>queueage</i> is invalid.
X'00000008'	X'00000240'	<i>queuetype</i> is invalid.
X'00000008'	X'00000250'	<i>count</i> is invalid.
X'00000008'	X'00000254'	<i>listaddress</i> is invalid.
X'00000008'	X'0000027C'	CQSQUERY FUNC=QNAME, CQSQUERY FUNC=QTYPE, or CQSQUERY FUNC=QOBS is not allowed for a resource structure.
X'00000008'	X'00000284'	Parmlist version is invalid.
X'00000008'	X'00000288'	List version is invalid.
X'0000000C'	X'00000300'	Request completed successfully for at least one, but not all, list entries. See <i>compcode</i> for individual errors.
X'0000000C'	X'00000304'	Request failed for all list entries. See <i>compcode</i> for individual errors.
X'00000010'	X'00000400'	A CQSRSYNC is required for this structure.
X'00000010'	X'00000404'	Structure inaccessible. Retry request later.
X'00000010'	X'00000430'	No CQS address space.
X'00000014'	X'00000500'	CQS internal error.

CQSREAD Request

Format for CQSREAD

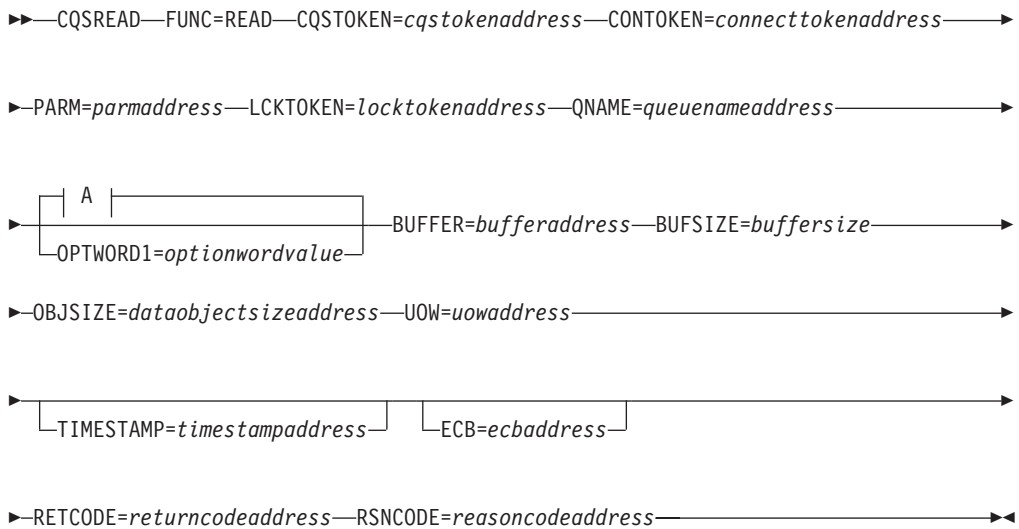
CONTINUE Function of CQSREAD: Use the CONTINUE function of a CQSREAD request to retrieve the rest of a data object after partial data is returned for a prior CQSREAD request.



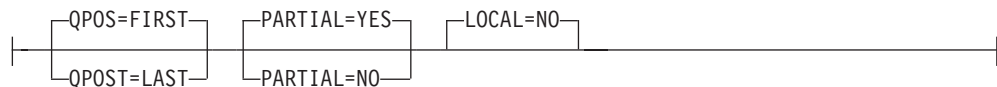
DSECT Function of CQSREAD: Use the DSECT function of a CQSREAD request to include equate (EQU) statements in your program for the CQSREAD parameter list length, CQSREAD return and reason codes, and literals that can be used to build the OPTWORD1 parameter.



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.



A:



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—▶▶
▶—PARAM=paramaddress—LCKTOKEN=locktokenaddress—QNAME=queueaddress—▶
▶—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—PARAM=paramaddress—LCKTOKEN=locktokenaddress—▶
▶—BUFFER=bufferaddress—BUFSIZE=buffersize—OBJSIZE=dataobjectsaddress—▶
▶—┌RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress—▶▶
  └ECB=ecbaddress—┘

```

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 159 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 z/OS 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=*dataobjectsizeaddress*

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 *dataobjectsize*.

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:

```
CQSREAD_PRTLNEQUX  PARTIAL=NO
CQSREAD_PRTLNEQUX  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:

```
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 135 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

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000004'	X'00000120'	The buffer size (<i>buffersize</i>) is less than the data object size (<i>dataobjectsize</i>). Partial data is returned. <i>dataobjectsize</i> contains the address of the actual data object size.
X'00000004'	X'00000124'	The buffer size (<i>buffersize</i>) is less than the data object size (<i>dataobjectsize</i>). No data is returned because PARTIAL=NO was specified. <i>dataobjectsize</i> contains the address of the actual data object size.
X'00000004'	X'00000128'	No data object to retrieve on queue name specified.
X'00000004'	X'0000012C'	No partial data to return.
X'00000008'	X'00000210'	<i>cqstoken</i> is invalid.
X'00000008'	X'00000214'	<i>connecttoken</i> is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'0000021C'	<i>locktoken</i> is invalid.
X'00000008'	X'00000220'	<i>queuname</i> is invalid.
X'00000008'	X'00000224'	<i>bufferaddress</i> is invalid.
X'00000008'	X'00000228'	<i>buffersize</i> is invalid.
X'00000008'	X'0000022C'	<i>dataobjectsize</i> is invalid.
X'00000008'	X'00000230'	<i>uow</i> is invalid.
X'00000008'	X'00000234'	Lock token address is invalid.
X'00000008'	X'00000278'	The request specified LOCAL=YES, but the requested object was placed on the structure using LOCAL=NO.
X'00000008'	X'0000027C'	CQSREAD is not allowed for a resource structure.
X'00000008'	X'00000284'	Parmlist version is invalid.
X'00000010'	X'00000400'	A CQSRSYNC is required for this structure.
X'00000010'	X'00000404'	Structure inaccessible. Retry request later.
X'00000010'	X'00000408'	Current position lost; cannot process CQSREAD FUNC=CONTINUE request.
X'00000010'	X'00000430'	No CQS address space.
X'00000010'	X'00000440'	Object lost because of rebuild.
X'00000014'	X'00000500'	CQS internal error.

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—FUNC=DELETE—CQSTOKEN=cqstokenaddress—————▶
▶—CONTOKEN=connecttokenaddress—PARM=parmaddress—————▶
▶—CLDTOKEN=coldqueuetokenaddress—UOW=uowaddressaddress————▶
└─ECB=ecbaddress┘
▶—RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress————▶▶

```

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.

```

▶▶—CQSRECVR—FUNC=DSECT—————▶▶

```

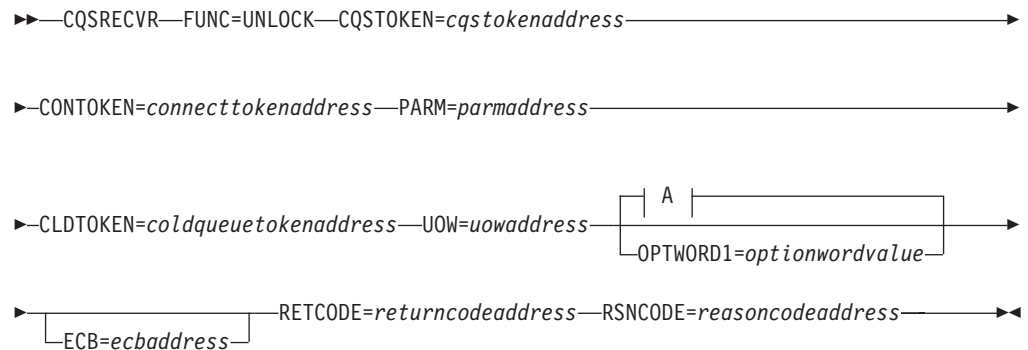
RETRIEVE Function of CQSRECVR: Use the RETRIEVE function of a CQSRECVR request to retrieve a copy of a data object associated with a UOW from the cold queue.

```

▶▶—CQSRECVR—FUNC=RETRIEVE—CQSTOKEN=cqstokenaddress————▶
▶—CONTOKEN=connecttokenaddress—PARM=parmaddress————▶
▶—CLDTOKEN=coldqueuetokenaddress—UOW=uowaddress—BUFFER=bufferaddress————▶
▶—BUFSIZE=buffer size—OBJSIZE=dataobjects sizeaddress————▶
└─ECB=ecbaddress┘
▶—RETCODE=returncodeaddress—RSNCODE=reasoncodeaddress————▶▶

```

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.

**A:****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 z/OS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise it is processed synchronously.

OBJSIZE=*dataobjectsizedataaddress*

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.

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.

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.

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_QPOSFEQUX  QPOS=FIRST
CQSRECVR_QPOSLEQUX  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 41. CQSRECVR Return and Reason Codes

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000004'	X'00000124'	<i>bufferize</i> is too small.
X'00000004'	X'00000128'	Data object for UOW not found on cold queue.
X'00000008'	X'00000210'	<i>cqstoken</i> is invalid.
X'00000008'	X'00000214'	<i>connecttoken</i> is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000224'	<i>bufferaddress</i> is invalid.
X'00000008'	X'00000228'	<i>bufferize</i> is invalid.

Table 41. CQSRECVR Return and Reason Codes (continued)

Return Code	Reason Code	Meaning
X'00000008'	X'0000022C'	<i>dataobjectsize</i> is invalid.
X'00000008'	X'00000230'	<i>uow</i> is invalid.
X'00000008'	X'00000234'	<i>coldqueuetoken</i> is invalid.
X'00000008'	X'0000027C'	CQSRECVR is not allowed for a resource structure.
X'00000008'	X'00000284'	Parmlist version is invalid.
X'00000010'	X'00000400'	A CQSRSYNC is required for this structure.
X'00000010'	X'00000404'	Structure is inaccessible. Retry request later.
X'00000010'	X'00000414'	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.
X'00000010'	X'00000430'	No CQS address space.
X'00000014'	X'00000500'	CQS internal error.

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.

▶▶—CQSREG—FUNC=DSECT—▶▶

REGISTER Function of CQSREG: Use the REGISTER function of a CQSREG request to register a client with a CQS.

▶▶—CQSREG—FUNC=REGISTER—PARAM=*parmaddress*—CQSSSN=*cqssystemnameaddress*—▶▶

▶—CLIENT=*clientnameaddress*—EVENT=*cqseventexit*—▶
 └─EVENTPARAM=*eventparmaddress*—┘

▶—CQSTOKEN=*cqstokenaddress*—VERSION=*cqsversionaddress*—▶▶

▶—RETCODE=*returncodeaddress*—RSNCODE=*reasoncodeaddress*—▶▶

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=*cqssystemnameaddress*

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'.

Return and Reason Codes for CQSREG

Table 42 on page 142 lists the return and reason code combinations that can be returned for CQSREG requests.

Table 42. CQSREG Return and Reason Codes

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000004'	X'00000100'	Client is already registered to CQS.
X'00000008'	X'00000244'	<i>clientname</i> is invalid.
X'00000008'	X'00000248'	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.
X'00000010'	X'0000040C'	CQS shutdown is pending.
X'00000010'	X'00000430'	The CQS address space is not active. The CQS address space must be started.
X'00000010'	X'00000438'	Another address space is already registered with CQS using the client ID (passed on a CQSREG request).
X'00000010'	X'00000440'	The user ID of the client address space is not authorized to register with this CQS.
X'00000014'	X'00000500'	CQS internal error.
X'00000014'	X'00000504'	Unable to obtain storage in client's address space for CQS's use.
X'00000014'	X'00000508'	Unable to obtain storage (CCIB).
X'00000014'	X'0000050C'	Unable to obtain storage (CRET).
X'00000014'	X'00000510'	CQS internal error (Loc ASCB).
X'00000014'	X'00000514'	Unable to establish z/OS Resource Manager routine to monitor CQS for the registering client.
X'00000014'	X'00000518'	CQS internal error (ESTAE add).
X'00000014'	X'0000051C'	CQS internal error (NmTkn Retrv).
X'00000014'	X'00000520'	CQS internal error (CGCT error).
X'00000014'	X'00000524'	CQS internal error (TTKN error).
X'00000014'	X'00000528'	CQS internal error (ALESERV error).
X'00000014'	X'0000052C'	CQS internal error (BPESVC error).
X'00000014'	X'00000530'	Unable to establish z/OS Resource Manager routine to monitor the client for CQS.
X'00000014'	X'00000534'	An abend occurred during CQSREG processing.

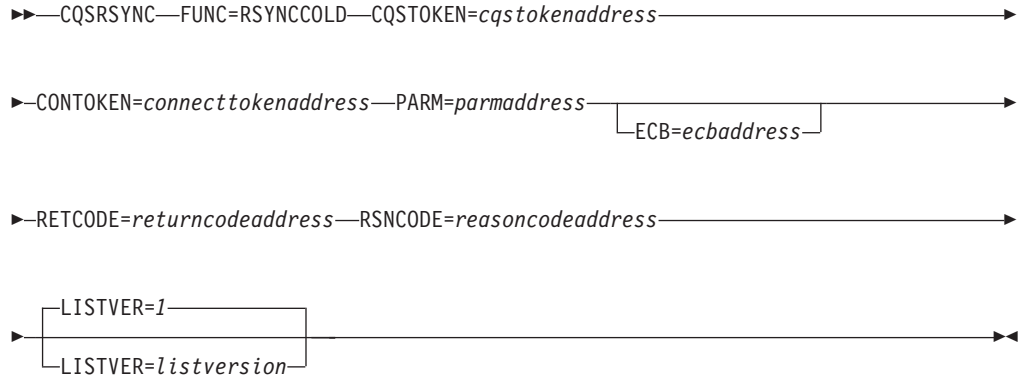
CQSRSYNC Request

Format for CQSRSYNC

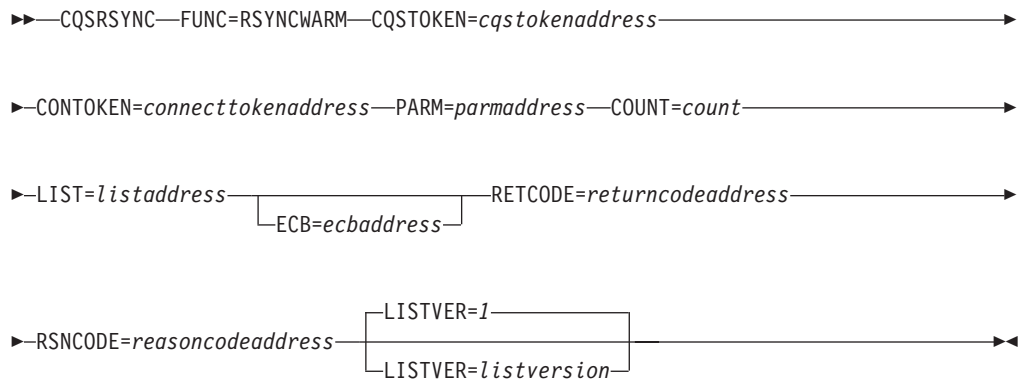
DSECT Function of CQSRSYNC: Use the DSECT function of a CQSRSYNC request to include equate (EQU) statements in your program for the CQSRSYNC parameter list length and CQSRSYNC return and reason codes.

▶▶—CQSRSYNC—FUNC=DSECT—◀◀

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.



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.



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 z/OS 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 43. UOW Status from the Client

Status	Meaning
X'0010'	Put Complete 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.
X'0020'	Read The data object for the UOW is assumed to be locked on the coupling facility.
X'0030'	Unlock 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.
X'0040'	Move 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.
X'0050'	Delete A CQSDEL request with lock token was issued for the UOW. The data object is assumed to have been deleted from the coupling facility.

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 44. UOW Status from CQS

Status	Meaning
X'0010'	Put Insync 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.

Table 44. UOW Status from CQS (continued)

Status	Meaning
X'0012'	Resync Deferred 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.
X'0020'	Read Insync 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.
X'0030'	Unlock Insync 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.
X'0050'	Delete Insync 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.
X'00F1'	Locked One of the following conditions exists: <ul style="list-style-type: none"> 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. 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. 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.
X'00F2'	Unknown Client status is any valid client status. The UOW is unknown to CQS. 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.

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

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully and all list entries are in sync. 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.

Table 45. CQSRSYNC Return and Reason Codes (continued)

Return Code	Reason Code	Meaning
X'00000004'	X'00000110'	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.
X'00000004'	X'00000114'	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.
X'00000004'	X'00000118'	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.
X'00000004'	X'0000011C'	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.
X'00000008'	X'00000210'	<i>cqstoken</i> is invalid. No list entries were processed. The Structure Event exit routine is not called. The client must reissue the CQSRSYNC request.
X'00000008'	X'00000214'	<i>connecttoken</i> is invalid. No list entries were processed. The Structure Event exit routine is not called. The client must reissue the CQSRSYNC request.
X'00000008'	X'00000218'	FUNC is invalid. The client must reissue the CQSRSYNC request.
X'00000008'	X'00000254'	<i>listaddress</i> is invalid. No list entries were processed. The Structure Event exit routine is not called. The client must reissue the CQSRSYNC request.
X'00000008'	X'0000027C'	CQSRSYNC is not allowed for a resource structure.
X'00000008'	X'00000284'	Parmlist version is invalid.
X'00000008'	X'00000288'	List version is invalid.
X'0000000C'	X'00000300'	Request succeeded for at least one, but not all, list entries. At least one list entry is in sync. See <i>compcode</i> 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.
X'0000000C'	X'00000304'	Request failed for all list entries. None of the list entries are in sync. See <i>compcode</i> 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.
X'00000010'	X'00000430'	No CQS address space.
X'00000014'	X'00000500'	CQS internal error.

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*—PARAM=*parmaddress*—▶▶

▶▶—ECB=*ecbaddress*—RETCODE=*returncodeaddress*—RSNCODE=*reasoncodeaddress*—▶▶

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 z/OS.

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 z/OS event control block (ECB) used for asynchronous requests. If ECB is specified, the request is processed asynchronously; otherwise, it is processed synchronously.

PARAM=*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 46. CQSSHUT Return and Reason Codes

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000008'	X'00000210'	<i>cqstoken</i> is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000284'	Parmlist version is invalid.
X'00000010'	X'00000444'	CQS initialization is in progress. Reissue the CQSSHUT request after initialization is complete.

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.

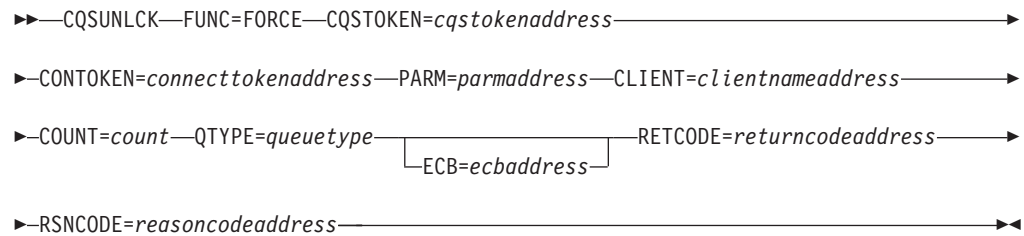
▶—CQSUNLCK—FUNC=UNLOCK—CQSTOKEN=*cqstokenaddress*—▶

▶—CONTOKEN=*connecttokenaddress*—PARAM=*parmaddress*—COUNT=*count*—▶

▶—LIST=*listaddress*—LISTVER=*listversion*—ECB=*ecbaddress*—▶

▶—RETCODE=*returncodeaddress*—RSNCODE=*reasoncodeaddress*—▶▶

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, resyncs 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. 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 insure 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.

It is up to the CQS clients in the IMSplex to insure that exactly one CQS client issues the CQSUNLCK FUNC=FORCE request successfully. If a CQS client issues the CQSUNLCK FUNC=FORCE request and a failure occurs, such as CQSUNLCK error, structure failure, loss of link, and so on, then the CQS clients in the IMSplex must insure that the CQSUNLCK FUNC=FORCE request is issued successfully after the error is corrected.

- 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 insure 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 insures 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 resyncs 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 z/OS 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 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

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000008'	X'00000210'	<i>cqstoken</i> is invalid.
X'00000008'	X'00000214'	<i>connecttoken</i> is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000240'	<i>queuetype</i> is invalid.
X'00000008'	X'00000244'	<i>clientname</i> is invalid.
X'00000008'	X'00000250'	<i>count</i> is invalid.
X'00000008'	X'00000254'	<i>listaddress</i> is invalid.

Table 47. CQSUNLCK Return and Reason Codes (continued)

Return Code	Reason Code	Meaning
X'00000008'	X'0000027C'	CQSUNLCK is not allowed for a resource structure.
X'00000008'	X'00000284'	Parmlist version is invalid.
X'00000008'	X'00000288'	List version is invalid.
X'0000000C'	X'00000300'	Request succeeded for at least one but not all list entries. See <i>compcode</i> for individual errors.
X'0000000C'	X'00000304'	Request failed for all list entries. See <i>compcode</i> for individual errors.
X'00000010'	X'00000400'	A CQSRSYNC is required for this structure.
X'00000010'	X'00000430'	No CQS address space.

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*—►►

►—CONTOKEN=*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 z/OS 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'

Resourcetype 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.

X'00000044'

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=parmaddress

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 4-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 4-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

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000008'	X'00000210'	<i>cqstoken</i> is invalid.
X'00000008'	X'00000214'	<i>contoken</i> is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000250'	<i>resourcelistcount</i> is invalid.
X'00000008'	X'00000254'	<i>listaddress</i> is invalid.
X'00000008'	X'00000280'	Request not allowed for a queue structure.
X'00000008'	X'00000284'	Parmlist version is invalid.
X'00000008'	X'00000288'	List version is invalid.
X'0000000C'	X'00000300'	Request succeeded for at least one but not all list entries. See <i>compcode</i> for individual errors.
X'0000000C'	X'00000304'	Request failed for all entries. See <i>compcode</i> 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 <i>resourcetype</i> 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 160 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 - GETDOBJ 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                                                 *
*                                                                 *
*****

```

Figure 25. Sample for CQSREAD Request (Part 1 of 4)

```

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 RDRRQTK,0(R4) SAVE CQS REGISTRATION TOKEN
MVC RDRCONTK,0(R5) SAVE CQS CONNECT TOKEN
ST R9,RDRECBA SAVE A(ECB)
LA R2,RDRPARM LOAD A(PARAMETER AREA) INTO R2
XC RDRLCKTK,RDRLCKTK 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=@(RDRRQTK), A(REGISTRATION TOKEN) X
PARM=(R2), A(CQSREAD PARMLIST AREA) X
CONTOKEN=@(RDRCONTK), A(CONNECT TOKEN) X
ECB=RDRECBA, A(ECB) X
LCKTOKEN=@(RDRLCKTK), 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
* OTHER - RETURN R0, R15 IN PARM LIST
LA R15,RC08 CQS 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
RDRRQTK  DS    XL16          CQS REGISTRATION TOKEN
RDRCONTK DS    XL16          CQS CONNECT TOKEN
RDRLCKTK DS    XL16          LOCKTOKEN (RETURNED)
RDRUOW   DS    XL32          UOW (RETURNED)

RDRQNAME DS    0XL16          QUEUE NAME
          DC    X'05'          CLIENT QUEUE TYPE 5
          DC    CL15'FFSTR01CF02CQ04'

RDROBJSZ DS    F             OBJECT SIZE (RETURNED)
RDRRSN   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)
RDRPARAM 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)

Chapter 7. CQS Client Exit Routines

This section describes the Common Queue Server (CQS) client exit routines.

In this section:

- “Client CQS Event Exit Routine”
- “CQS Client Structure Event Exit Routine” on page 167
- “CQS Client Structure Inform Exit Routine” on page 175

This section 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 interrupt request block (IRB) under the client task control block (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 can 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 might 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

Register	Contents
0	Length in bytes of the parameter list pointed to by R1.
1	Address of CQS Event Exit Parameter List (mapped by macro CQSCEVX).

- 13** Address of a standard 18-word save area, immediately followed by 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 might have changed, or it might 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).

Register	Contents
13	The same value it had on entry to the CQS Event exit routine.
15	Return code
0	Always set this to zero.

CQS Restart Entry Parameter List

Table 49 describes the CQS restart entry parameters for the Client CQS Event exit routine.

Table 49. Client CQS Event Exit Routine Parameter List: CQS Restart Entry

Field Name	Offset	Length	Description
CEVX_PVSN	X'00'	X'04'	Parameter List Version Number (00000001).
CEVX_EVENT	X'04'	X'04'	CQS Event Code 1 CQS Initialization Event (CEVX_INIT).
CEVX_SCODE	X'08'	X'04'	CQS Event Subcode 1 Client can re-register and reconnect to CQS (CEVX_RESTART).
CEVX_DATA	X'0C'	X'04'	Event exit routine client data that was passed to CQS on the CQSREG request.
CEVX_CQSID	X'10'	X'08'	CQS identifier.
CEVX_CQSVER	X'18'	X'04'	CQS version number.
CEVX_TSTMP	X'1C'	X'08'	Time stamp representing the time the exit routine was scheduled (in STCK format).

CQS Abnormal Termination Parameter List

Table 50 on page 167 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

Offset	Length	Description
X'00'	X'04'	Parameter List Version Number (00000001).
X'04'	X'04'	CQS Event Code 2 CQS Termination Event.
X'08'	X'04'	CQS Event Subcode 1 CQS abnormal termination entry. The CQS address space is terminating abnormally.
X'0C'	X'04'	Event exit routine client data that was passed to CQS on the CQSREG request.
X'10'	X'08'	CQS identifier.
X'18'	X'04'	CQS version number.
X'1C'	X'08'	Time stamp representing the time the exit routine was scheduled (in STCK format).
X'24'	X'04'	Abnormal Termination reason code. (CQS abend code)

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 might 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.

- **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

Register	Contents
0	Length in bytes of the parameter list pointed to by R1.
1	Address of Client Structure Event exit routine parameter list (mapped by macro CQSSEVX).
13	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.
14	Return address.
15	Entry point of exit routine.

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 might have changed, or it might 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.

Register	Contents
----------	----------

- 13 The same value it had on entry to the Client Structure Event exit routine.
- 15 Return code
 - 0 Always set this to zero.

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 51. Client Structure Event Exit Routine Parameter List: Deferred Resync Complete

Offset	Length	Description
X'00'	X'04'	Parameter List Version Number (00000001).
X'04'	X'04'	Structure Event Code 1 Resync UOW Event.
X'08'	X'04'	Structure Event Subcode 1 Deferred Resync complete.
X'0C'	X'04'	Structure Event exit routine client data that was passed to CQS on the CQSCONN request.
X'10'	X'08'	CQS identifier.
X'18'	X'04'	CQS version number.
X'1C'	X'10'	Structure Name.
X'2C'	X'08'	Time stamp representing the time the exit routine was scheduled (in STCK format).
X'34'	X'20'	Unit of work (UOW) identifier.
X'54'	X'10'	Queue Name.
X'64'	X'10'	Deferred Resync token. This is the Put token that is used for Put Forget processing.
X'74'	X'02'	CQS UOW State X'0010' Put Insync 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. X'00F2' Unknown 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.
X'76'	X'02'	Reserved.

CQS Resync Parameter List

Table 52 on page 170 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

Offset	Length	Description
X'00'	X'04'	Parameter List Version Number (00000001).
X'04'	X'04'	Structure Event Code 1 Resync UOW Event.
X'08'	X'04'	Structure Event Subcode 2 CQS Initiated Resync processing.
X'0C'	X'04'	Structure Event exit routine client data that was passed to CQS on the CQSCONN request.
X'10'	X'08'	CQS identifier.
X'18'	X'04'	CQS version number.
X'1C'	X'10'	Structure Name.
X'2C'	X'08'	Time stamp representing the time the exit routine was scheduled (in STCK format).
X'34'	X'04'	Number of unit of work (UOW) list entries.
X'38'	X'04'	Length of each UOW list entry.
X'3C'	X'04'	Offset into parmlist of start of UOW list. The parmlist is one contiguous piece of storage, including the UOW list.

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

Offset	Length	Description
X'00'	X'20'	Unit of work (UOW) identifier.
X'20'	X'10'	Queue name.
X'30'	X'10'	Resync token. <ul style="list-style-type: none"> 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)

Offset	Length	Description
X'40'	X'02'	CQS UOW Status
		X'00F1' 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.
		X'00F3' 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.
		X'00F4' 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.
X'42'	X'02'	Reserved.

Checkpoint Parameter List for CQS Client Structure Event

Table 54 describes the checkpoint parameters for the Client Structure Event exit routine.

Table 54. Client Structure Event Exit Routine Parameter List: Checkpoint

Offset	Length	Description
X'00'	X'04'	Parameter List Version Number (00000001).
X'04'	X'04'	Structure Event Code
		2 Checkpoint Event.
X'08'	X'04'	Structure Event Subcode
		1 Structure checkpoint begin.
		2 Structure checkpoint end.
		3 Structure checkpoint failure.
X'0C'	X'04'	Structure Event exit routine client data that was passed to CQS on the CQSCONN request.
X'10'	X'08'	CQS identifier.
X'18'	X'04'	CQS version number.
X'1C'	X'10'	Structure Name.
X'2C'	X'08'	Time stamp representing the time the exit routine was scheduled (in STCK format).
X'34'	X'08'	CQS identifier of the master CQS performing the checkpoint process.

Table 54. Client Structure Event Exit Routine Parameter List: Checkpoint (continued)

Offset	Length	Description
X'3C'	X'01'	Flag byte.
		X'80' This CQS is the master of the process. The CQS identifier and master CQS identifier are the same.
X'3D'	X'03'	Reserved.

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

Offset	Length	Description
X'00'	X'04'	Parameter List Version Number (00000001).
X'04'	X'04'	Structure Event Code
		3 Structure Rebuild Event.
X'08'	X'04'	Structure Event Subcode
		1 Structure rebuild begin.
		2 Structure rebuild (copy) end.
		3 Structure rebuild (copy) failure.
		4 Structure rebuild failure.
		5 Structure rebuild (recovery) end.
		6 Structure rebuild (recovery) failure.
X'0C'	X'04'	Structure Event exit routine client data that was passed to CQS on the CQSCONN request.
X'10'	X'08'	CQS identifier.
X'18'	X'04'	CQS version number.
X'1C'	X'10'	Structure Name.
X'2C'	X'08'	Time stamp representing the time the exit routine was scheduled (in STCK format).
X'34'	X'08'	CQS identifier of the master CQS performing the rebuild process.
X'3C'	X'01'	Flag byte.
		X'80' This CQS is the master of the process. The CQS identifier and master CQS identifier are the same.
X'3D'	X'03'	Reserved.

Structure Rebuild Lost UOWs Parameter List for CQS Client Structure Event

Table 56 on page 173 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 might 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

Offset	Length	Description
X'00'	X'04'	Parameter List Version Number (00000001).
X'04'	X'04'	Structure Event Code 3 Structure Rebuild Event.
X'08'	X'04'	Structure Event Subcode 7 Structure recovery lost UOWs. Important: This subcode applies only to queue structures.
X'0C'	X'04'	Structure Event exit routine client data that was passed to CQS on the CQSCONN request.
X'10'	X'08'	CQS identifier.
X'18'	X'04'	CQS version number.
X'1C'	X'10'	Structure Name.
X'2C'	X'08'	Time stamp representing the time the exit routine was scheduled (in STCK format).
X'34'	X'08'	CQS identifier of the master CQS performing the rebuild process.
X'3C'	X'01'	Flag byte. X'80' This CQS is the master of the process. The CQS identifier and master CQS identifier are the same.
X'3D'	X'03'	Reserved.
X'40'	X'04'	Number of Lost UOW list entries.
X'44'	X'04'	Length of each Lost UOW list entry.
X'48'	X'04'	Offset into parmlist of start of Lost UOW list. The parmlist is one contiguous piece of storage, including the Lost UOW list.

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

Offset	Length	Description
X'00'	X'20'	Unit of work (UOW) identifier.
X'20'	X'10'	Client Queue Name.
X'30'	X'1'	Lost UOW status. X'80' Lost UOW was on client queue. X'40' Lost UOW was locked. X'20' Lost UOW was on COLDQ. X'10' Lost UOW was on CQS private queue.
X'31'	X'3'	Reserved.

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

Offset	Length	Description
X'00'	X'04'	Parameter List Version Number (00000001).
X'04'	X'04'	Structure Event Code 4 Structure Overflow Event.
X'08'	X'04'	Structure Event Subcode 1 Move queues to overflow. One or more queues was selected as candidates to be moved to the overflow structure and was approved by the Queue Overflow user exit routine. 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.
X'0C'	X'04'	Structure Event exit routine client data that was passed to CQS on the CQSCONN request.
X'10'	X'08'	CQS identifier.
X'18'	X'04'	CQS version number.
X'1C'	X'10'	Structure Name.
X'2C'	X'08'	Time stamp representing the time the exit routine was scheduled (in STCK format).
X'34'	X'08'	CQS identifier of the master CQS performing the overflow process.
X'3C'	X'01'	Flag byte. X'80' This CQS is the master of the process. The CQS identifier and master CQS identifier are the same. X'40' The structure is no longer in overflow mode. This value applies only to subcode 2.
X'3D'	X'03'	Reserved.
X'40'	X'04'	Number of Queue Name entries in the list.
X'44'	X'04'	Length of each Queue Name list entry.
X'48'	X'04'	Offset into parmlist of start of Queue Name list. Each Queue Name list entry contains the 16-byte queue name of a queue that is being moved to the overflow structure. The parmlist is one contiguous piece of storage, including the Queue Name list.

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

Offset	Length	Description
X'00'	X'04'	Parameter List Version Number (00000002).

Table 59. Client Structure Event Exit Routine Parameter List: Structure Status Change (continued)

Offset	Length	Description
X'04'	X'04'	Structure Event Code. 5 Structure Status Change Event.
X'08'	X'04'	Structure Event Subcode 1 Structure available again after a loss. 2 The structure failed. 3 CQS lost its connection to the structure (STXLCONN). 4 The log stream is becoming available, making the structure available (STXAVLOG). Important: This subcode applies only to queue structures. 5 The log stream is becoming unavailable, making the structure unavailable (STXFLOG). Important: This subcode applies only to queue structures. 6 Structure repopulation required due to structure failure.
X'0C'	X'04'	Structure Event exit routine client data that was passed to CQS on the CQSCONN request.
X'10'	X'08'	CQS identifier.
X'18'	X'04'	CQS version number.
X'1C'	X'10'	Structure Name.
X'2C'	X'08'	Time stamp representing the time the exit routine was scheduled (in STCK format).
X'34'	X'01'	Structure type 1 Queue structure 2 Resource structure
X'38'	X'18'	Not used.
X'50'	X'08'	Structure version of new structure that requires repopulation, because old structure failed.

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).

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

Register	Contents
0	Length in bytes of the parameter list pointed to by R1.
1	Address of CQS Structure Inform Exit Parameter List (mapped by macro CQSINFX).
13	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.
14	Return address.
15	Entry point of exit routine.

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 might have changed, or it might 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 might also use the 18-word area following the save area for additional save area or work area storage.

Register	Contents
13	Same value as it had on entry to the CQS Structure Inform exit routine.
15	Return code
0	Always set this to zero.

Structure Inform Parameter List for CQS Client Structure Inform

Table 60 describes the parameters for the Client Structure Inform exit routine.

Table 60. Client Structure Inform Exit Routine Parameter List

Offset	Length	Description
X'00'	X'04'	Parameter List Version Number (00000001).
X'04'	X'04'	Structure Inform exit routine client data that was passed to CQS on the CQSCONN request.
X'08'	X'08'	CQS identifier.
X'10'	X'04'	CQS version number.
X'14'	X'10'	Structure Name.

Table 60. Client Structure Inform Exit Routine Parameter List (continued)

Offset	Length	Description
X'24'	X'08'	Time stamp representing the time the exit routine was scheduled (in STCK format).
X'2C'	X'04'	Number of Queue Names entries in the list.
X'30'	X'04'	Length of each Queue Name list entry.
X'34'	X'04'	Offset into parmlist 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 parmlist is one contiguous piece of storage, including the Queue Name list.

Chapter 8. CQS Diagnosis

This section describes diagnostic information that helps you analyze problems in CQS.

In this section:

“CQS Log Records”

“Printing CQS Log Records” on page 181

CQS Log Records

CQS writes records to the z/OS log stream that contains all CQS log records from all CQs 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 61. CQS Log Records

Type	Subtype	Mapping Macro	Conditions for Writing the Log Record
X'03'	X'01'	CQSLGCON	CQSCONN request: The client connect to a structure completed.
X'04'	X'01'	CQSLGDSC	CQSDISC request: The client disconnect from a structure completed.
X'07'	X'01'	CQSLGPUT	CQSPUT OBJECT request completed.
	X'02'		CQSPUT COMMIT request completed.
	X'03'		CQSPUT START request completed.
	X'04'		CQSPUT FORGET request completed.
	X'05'		CQSPUT ABORT request completed.
	X'06'		CQSPUT request failed.
	X'07'		CQSPUT system checkpoint record was written.
	X'08'		CQSPUT FORGET request completed. This is a batched log record.
X'08'	X'01'	CQSLGRD	CQSREAD request completed.
	X'02'		CQSREAD request failed.
	X'03'		CQSREAD system checkpoint record was written.

Table 61. CQS Log Records (continued)

Type	Subtype	Mapping Macro	Conditions for Writing the Log Record
		CQSLGCHD	This system checkpoint header record is not a complete log record, but it is used in CQSLGPUT and CQSLGRD system checkpoint log records.
X'0B'	X'01' X'02' X'03'	CQSLGMOV	CQSMOVE or CQSUNLCK request completed. CQSMOVE or CQSUNLCK request failed. CQSMOVE or CQSUNLCK request moved an object between the primary and overflow structure.
X'0D'	X'01' X'02' X'03' X'04'	CQSLGDEL	CQSDDEL request: Delete-type 1 (delete by token) completed. CQSDDEL request: Delete-type 2 (delete by queue name) completed. CQSDDEL request: Delete-type 3 (delete by queue name and UOW) completed. CQSDDEL request: Delete-type 1 (delete by token) completed. This is a batched log record.
		CQSLGBHD	This batched log record header record is not a complete log record, but is used in CQSLGPUT and CQSLGDEL batched log records.
X'10'	X'01'	CQSLGSHT	CQSSHUT request completed.
X'32'	X'01' X'02' X'03'	CQSLGYCH	System checkpoint started. System checkpoint ended. System checkpoint failed.
X'40'	X'01'	CQSLGIST	Beginning of log stream.
X'42'	X'01' X'02' X'03'	CQSLGTCH	Structure checkpoint started. Structure checkpoint ended. Structure checkpoint failed.
X'43'	X'01'	CQSLGRBL	Structure rebuild started. Statistics about the old structure, the rebuild structure, and rebuild failure are mapped by CQSSSTT6.
	X'02'		Structure rebuild ended. Statistics about the old structure, the rebuild structure, and rebuild failure are mapped by CQSSSTT6.
	X'03'		Structure rebuild failed. Statistics about the old structure, the rebuild structure, and rebuild failure are mapped by CQSSSTT6.
	X'04'		Structure rebuild resulted in a lost UOW list. This record lists the lost UOWs.
X'44'	X'01' X'02' X'03' X'04' X'06' X'07' X'08'	CQSLGOFL	Overflow threshold began. Overflow threshold ended. Overflow threshold failed. Overflow mode ended. Qnames were moved to overflow. Qnames were removed from overflow. CQSOVERFLOWQNMR, a control list entry containing the list of queue names deleted from overflow, was deleted.
	X'09'		Overflow Scan Begin.
	X'0A'		Overflow Scan End.
	X'0B'		Private Queue Scan Begin.
	X'0C'		Structure to be deleted.

Table 61. CQS Log Records (continued)

Type	Subtype	Mapping Macro	Conditions for Writing the Log Record
X'60'	X'01'	CQSLGSTT BPESSTA	Structure statistics were written at the end of system checkpoint. Internal BPE service statistics were also written at the end of system checkpoint.
	X'C0'		

Printing CQS Log Records

To print the CQS log records from the z/OS 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 a z/OS system log. This JCL causes the z/OS logger to invoke the default log stream subsystem exit routine, IXGSEXIT, to copy the log records. The exit routine returns a maximum of 32 760 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:

```
//CQSERA10 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 EXITR=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 32 760 bytes (X'7FF8')).

Related Reading: For a complete description of the IMS File Select and Formatting Print utility, see *IMS Version 9: Utilities Reference: System*.

For a complete description of the z/OS logger subsystem exit (IXGSEXIT) usage and parameters, see *OS/390 MVS Diagnosis: 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 service 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 service. The copy made by IEBGENER is a binary image of the log records. The JCL in Figure 29 on page 183 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:


```

//CQSCPYLG JOB MSGLEVEL=1,CLASS=K
//*****
//* THIS JOB COPIES A CQS LOG STREAM TO A DATASET (MAX 32K / RECORD) *
//*****
//STEP1 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD DUMMY
//SYSUT1 DD DSN=SYSLOG.MSGQ01.LOG,
//          SUBSYS=(LOGR,IXGSEXIT,
//          'FROM=(2001/089,15:10),TO=(2001/089,15:30),LOCAL'),
//          DCB=(BLKSIZE=32760)
//SYSUT2 DD DSN=CQS.LOG.COPY,
//          DISP=(NEW,KEEP,DELETE),
//          VOL=SER=EDSDMP,
//          SPACE=(CYL,(10,10)),
//          UNIT=SYSDA

```

Figure 29. JCL to Copy CQS Records from a Specific Time Period

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 32K bytes in length will be truncated. The SUBSYS exit supports a maximum of a 32K 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 9: 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 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 section or by the following marking: Diagnosis, Modification or Tuning Information....

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

BookManager	OS/390
CICS	Parallel Sysplex
IBM	Processor Resource/Systems Manager
IMS	PR/SM
Language Environment	RACF
MVS	S/390
MVS/DFP	System/390
MVS/ESA	z/OS

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 9 library.

- *External Security Interface (RACROUTE) Macro Reference*, GC28-1366
- *z/OS DFSMS Access Method Services for Catalogs*, SC26-7394
- *z/OS MVS Initialization and Tuning Reference*, SA22-7592
- *MVS Programming: Authorized Assembler Services Guide*, SA22-7608
- *MVS Programming: Sysplex Services Guide*, SA22-7617
- *z/OS MVS System Commands*, SA22-7627
- *z/OS MVS Programming: Assembler Services Guide*, SA22-7605
- *z/OS MVS Setting Up a Sysplex*, SA22-7625
- *System/390® MVS: Sysplex Hardware and Software Migration*, GC28-1210

IMS Version 9 Library

ZES1-2330	ADB	<i>IMS Version 9: Administration Guide: Database Manager</i>
ZES1-2331	AS	<i>IMS Version 9: Administration Guide: System</i>
ZES1-2332	ATM	<i>IMS Version 9: Administration Guide: Transaction Manager</i>
ZES1-2333	APDB	<i>IMS Version 9: Application Programming: Database Manager</i>
ZES1-2334	APDG	<i>IMS Version 9: Application Programming: Design Guide</i>
ZES1-2335	APCICS	<i>IMS Version 9: Application Programming: EXEC DLI Commands for CICS and IMS</i>
ZES1-2336	APTM	<i>IMS Version 9: Application Programming: Transaction Manager</i>
ZES1-2337	BPE	<i>IMS Version 9: Base Primitive Environment Guide and Reference</i>
ZES1-2338	CR	<i>IMS Version 9: Command Reference</i>
ZES1-2339	CQS	<i>IMS Version 9: Common Queue Server Guide and Reference</i>
ZES1-2340	CSL	<i>IMS Version 9: Common Service Layer Guide and Reference</i>

ZES1-2341	CG	<i>IMS Version 9: Customization Guide</i>
ZES1-2342	DBRC	<i>IMS Version 9: DBRC Guide and Reference</i>
ZES1-2343	DGR	<i>IMS Version 9: Diagnosis Guide and Reference</i>
ZES1-2344	FAST	<i>IMS Version 9: Failure Analysis Structure Tables (FAST) for Dump Analysis</i>
ZES1-2346	OLR	<i>IMS Version 9: HALDB Online Reorganization Guide</i>
ZES1-2380	CT	<i>IMS Version 9: IMS Connect Guide and Reference</i>
ZES1-2347	JGR	<i>IMS Version 9: IMS Java Guide and Reference</i>
ZES1-2348	IIV	<i>IMS Version 9: Installation Volume 1: Installation Verification</i>
ZES1-2349	ISDT	<i>IMS Version 9: Installation Volume 2: System Definition and Tailoring</i>
ZES1-2350	INTRO	<i>IMS Version 9: An Introduction to IMS</i>
ZES1-2351	MIG	<i>IMS Version 9: Master Index and Glossary</i>
ZES1-2352	MC1	<i>IMS Version 9: Messages and Codes, Volume 1</i>
ZES1-2353	MC2	<i>IMS Version 9: Messages and Codes, Volume 2</i>
ZES1-2354	OTMA	<i>IMS Version 9: Open Transaction Manager Access Guide and Reference</i>
ZES1-2355	OG	<i>IMS Version 9: Operations Guide</i>
GC17-7831	RPG	<i>IMS Version 9: Release Planning Guide</i>
ZES1-2358	URDBTM	<i>IMS Version 9: Utilities Reference: Database and Transaction Manager</i>
ZES1-2359	URS	<i>IMS Version 9: Utilities Reference: System</i>

Supplementary Publications

GC17-7825	LPS	<i>IMS Version 9: Licensed Program Specifications</i>
ZES1-2357	SOC	<i>IMS Version 9: Summary of Operator Commands</i>

Publication Collections

LK3T-7213	CD	IMS Version 9 Softcopy Library
LK3T-7144	CD	IMS Favorites

Publication Collections

LBOF-7789	Hardcopy and CD	Licensed Bill of Forms (LBOF): IMS Version 9 Hardcopy and Softcopy Library
SBOF-7790	Hardcopy	Unlicensed Bill of Forms (SBOF): IMS Version 9 Unlicensed Hardcopy Library
SK2T-6700	CD	OS/390 Collection
SK3T-4270	CD	z/OS Software Products Collection
SK3T-4271	DVD	z/OS and Software Products DVD Collection

Accessibility Titles Cited in this Book

SA22-7787		z/OS V1R1.0 TSO Primer
SA22-7794		z/OS V1R1.0 TSO/E User's Guide
SC34-4822		z/OS V1R1.0 ISPF User's Guide, Volume 1

Index

A

- abnormal termination or restart, client processing
 - after 167
- altering structures 34
- assembling a client program 77
- authorization
 - requests 70
- authorizing CQS
 - connections 33
 - registration 33
- Automatic Restart Manager
 - policy 32
 - using 31

B

- benefits of using CQS 2
- BPE (Base Primitive Environment)
 - common user exit routine execution environment 47
 - defining 12
- BPE Statistics user exit 67

C

- CFRM (coupling facility resource management)
 - couple data set format utility 43
 - policy 8
 - policy, defining 12, 34
- CFSizer 12
- checkpoint
 - client initiating 78
 - data set 35
 - structure, initiating 35
 - system, initiating 34
- cleanup failure 32
- client
 - exit routines (CQS) 165
 - Event 165
 - Structure Event 167
 - Structure Inform 175
 - interface
 - authorized 70
 - non-authorized 70
 - queue type 3
 - requests 6
- Client Connection user-supplied exit routine, CQS 49
- client program
 - assembling 77
 - writing 69
- client requests
 - assembling a program 77
 - authorization 70
 - coding 70
 - CQSBrowse 80
 - CQSCHKPT 87
 - CQSDEL 96
 - CQSDEREG 100

- client requests (*continued*)
 - CQSDISC 101
 - CQSINFRM 106
 - CQSMOVE 110
 - CQSPUT 114
 - CQSQUERY 121
 - CQSREAD 130
 - CQSRECVR 135
 - CQSREG 140
 - CQSRSYNC 142
 - CQSSHUT 149
 - CQSUNLCK 150
 - CQSUPD 155
 - ECB, using 74
 - environmental requirements 71
 - introduction 69
 - lists, using 75
 - literals, coding 74
 - parameters, coding 73
 - requests
 - CQSCONN 90
 - return and reason codes 75
 - sequence of 70
- Client Structure Event exit 167
 - parameters 169
- Client Structure Inform exit 175
 - parameters 176
- coding requests 70
- cold start 31
 - structures 29
- components of a CQS 2
- copy
 - structures 42
- copying log records 182
- couple data set format utility 43
- CQS (Common Queue Server) 7
 - administering 27
 - authorization 33
 - benefits 2
 - automatic work load balancing 2
 - incremental growth 2
 - reliability 2
 - client connection
 - establishing 32
 - client exit routines
 - Event 165
 - Structure Event 167
 - Structure Inform 175
 - client failure 32
 - clients 79
 - components 2
 - checkpoint data set 2
 - overflow structure 2
 - primary structure 2
 - resource structure 2
 - structure recovery data set 3
 - z/OS log stream 2
 - customizing 13

- CQS (Common Queue Server) *(continued)*
 - data sets 24
 - defining 7, 12
 - diagnosis 179
 - diagram of client systems and coupling facility 1
 - execution data sets 24
 - structure recovery data set 25
 - system checkpoint data set 24
 - execution parameters
 - specifying 14
 - exit routines.
 - See CQS user-supplied exit routines
 - failure 32
 - functions 3
 - overflow processing 3
 - records restart 3
 - requests 3
 - structure checkpoint 3
 - structure rebuild 3
 - system checkpoint 3
 - global structure definition PROCLIB member
 - keywords 19
 - information for restart 28
 - initialization parameters PROCLIB member
 - specifying 16
 - JCL for printing log records 181
 - local structure definition PROCLIB member
 - specifying 17
 - log records 179
 - logging 6
 - monitoring 13
 - multiple clients 11
 - notification of work 3
 - operating system requirements 1
 - overview 1
 - parameters
 - CQS PROCLIB 16
 - execution 14
 - preparing to start 12
 - printing log records 181
 - rebuilding structures 39
 - recovering 39
 - restarting 32
 - cold start 31
 - description 30
 - warm start 30
 - restarting information 28
 - restarting structures 28
 - allocation 28
 - restating after system checkpoint 35
 - shutting down 44
 - starting 27
 - starting manually 27
 - structure cold start 29
 - structure overflow function 37
 - structure types managed 3
 - structure warm start 28
 - tailoring 7
- CQS Event exit 165
 - abnormal termination 167
 - parameters 166
- CQS Event exit *(continued)*
 - parameters, abnormal termination 166
- CQS statistics
 - using BPE Statistics user exit 67
- CQS user-supplied exit routine
 - writing in assembler 48
- CQS user-supplied exit routines 47
 - Client Connection
 - general 49
 - parameters 50
 - register contents 49
 - general information 47
 - Initialization-Termination (Init-Term)
 - general 48
 - parameters 49
 - register contents 48
 - Queue Overflow
 - general 51
 - parameters 52
 - register contents 52
 - Structure Event
 - checkpoint parameters 64
 - connection parameters 63
 - general 62
 - overflow parameters 66
 - rebuild parameters 65
 - register contents 63
 - routine parameters 63
 - status change parameters 67
 - Structure Statistics
 - CQS request statistics record 55
 - data object statistics record 56
 - general 53
 - parameters 54
 - queue name statistics record 57
 - register contents 53
 - structure checkpoint statistics entry 61
 - structure checkpoint statistics record 60
 - structure process statistics record 55
 - structure rebuild statistics record 58
 - z/OS request statistics record 57
- CQS-managed rebuild 40
- CQSBRWSE request 80
 - BROWSE function 80
 - BRWSOBS function 80
 - COMPLETE function 80
 - CONTINUE function 81
 - DSECT function 81
 - functions 80
 - parameters 82
 - return and reason codes 86
 - syntax 80
 - usage 81
- CQSCHKPT request
 - CHKPTSTR function 87
 - CHKPTSYS function 88
 - DSECT function 88
 - format 87
 - parameters 89
 - return and reason codes 90
 - syntax 87

- CQSCHKPT request *(continued)*
 - usage 88
- CQSCONN request
 - CONNECT function 90
 - DSECT function 91
 - format 90
 - parameters 91
 - restrictions 91
 - return and reason codes 95
 - syntax 90
 - usage 91
- CQSDEL request
 - DELETE function 96
 - DSECT function 96
 - format 96
 - parameter 96
 - return and reason codes 99
 - syntax 96
 - usage 96
- CQSDEREG request
 - DEREGISTER function 100
 - DSECT function 100
 - format 100
 - parameters 100
 - return and reason codes 101
 - syntax 100
 - usage 100
- CQSDISC request
 - DISCABND function 101
 - DISCNORM function 102
 - DSECT function 102
 - format 101
 - parameters 103
 - return and reason codes 105
 - syntax 101
 - usage 102
- CQSINFRM request
 - DSECT function 106
 - format 106
 - INFORM function 106
 - parameters 107
 - return and reason codes 110
 - syntax 106
 - UNINFORM function 106
 - usage 107
- CQSIPxxx
 - format rules 16
 - overview 16
 - sample PROCLIB member 16
- CQSMOVE request
 - DSECT function 110
 - format 110
 - MOVE function 110
 - parameters 112
 - return and reason codes 113
 - syntax 110
 - usage 111
- CQSPUT request
 - ABORT function 114
 - actions 116
 - DSECT function 114
- CQSPUT request *(continued)*
 - FORGET function 114
 - format 114
 - parameters 117
 - PUT function 114
 - return and reason codes 120
 - syntax 114
 - usage 115
- CQSQUERY request
 - DSECT function 121
 - format 121
 - parameters 125
 - QNAME function 121
 - QRYOBS function 122
 - QTYPE function 122
 - return and reason codes 129
 - STATISTICS function 123
 - STRSTAT function 123
 - syntax 121
 - usage 123
- CQSREAD request
 - CONTINUE function 130
 - DSECT function 130
 - example 159
 - format 130
 - functions 130
 - parameters 132
 - READ function 130
 - REREAD function 131
 - return and reason codes 134
 - syntax 130
 - usage 131
- CQSRECVR request
 - DELETE function 135
 - DSECT function 136
 - format 135
 - functions 135
 - parameters 137
 - RETRIEVE function 136
 - return and reason codes 139
 - syntax 135
 - UNLOCK function 136
 - usage 137
- CQSREG request
 - DSECT function 140
 - functions 140
 - parameters 140
 - REGISTER function 140
 - return and reason codes 141
 - syntax 140
 - usage 140
- CQSRSYNC request
 - DSECT function 142
 - format 142
 - functions 142
 - parameters 144
 - return and reason codes 147
 - RSYNCCOLD function 143
 - RSYNCWARM function 143
 - syntax 142
 - usage 143

CQSSGxxx
 formatting rules 19
 overview 19
 sample PROCLIB member 20

CQSSHUT request
 DSECT function 149
 format 149
 functions 149
 parameters 149
 QUIESCE function 149
 return and reason codes 150
 syntax 149
 usage 149

CQSSLxxx
 formatting rules 18
 overview 17
 sample PROCLIB member 18

CQSUNLCK request
 DSECT function 150
 FORCE function 150
 format 150
 functions 150
 parameters 152
 return and reason codes 154
 syntax 150
 UNLOCK function 150
 usage 151

CQSUPD request
 DSECT function 155
 format 155
 functions 155
 parameters 155
 return and reason codes 159
 syntax 155
 UPDATE function 155
 usage 155

cross-system coupling facility 14

D

data sets
 CQS execution 24
 entry-sequenced 24
 IMS.ADFSMAC 77
 structure recovery 25
 system checkpoint 24

defining
 BPE 12
 CQS 12
 policies 8
 z/OS policies 7

deleting structures 44

DFSERA10 181

diagnosis 179
 CQS log records 179
 printing log records 181

display for structure full threshold 38

duplexing
 explicitly stopping 43
 structure 5, 42
 unnecessary overhead 43

E

ECB (z/OS event control block), using with client request 74

EMHQ
 disabling 20

EMHQ structures 8
 disabling 18

ENF 40

entry-sequenced data set 24

environment
 CQS deregister request 72
 CQS register request 72
 CQS requests, authorized interface 71
 CQS requests, non-authorized interface 71

environments
 client requests 71

ESDS 24

event notification facility 40

events, handling 77
 cold start 77
 registering interest in queues 78
 shutting down CQS 78
 tuning for performance 78

example
 coding CQSREAD with OPTWORD1 74
 CQSIPxxx PROCLIB member 16
 CQSREAD request 159
 CQSSGxxx PROCLIB member 20
 CQSSLxxx sample PROCLIB member 18
 DD card to add local time and date 182
 DD card to limit log records printed 182
 defining IMS resources
 in the CFRM policy 11
 in the LOGR policy 11
 in the SFM policy 11
 display for structure full threshold 38
 explicitly stopping duplexing 43
 JCL to copy CQS records from specific time period 183
 JCL to print CQS log records 181
 OBJAVGSZ calculation 22
 passing a value
 for register 73
 for symbol 73
 for symbol value 74
 passing an address
 for register 73
 for symbol 73
 passing an equate for symbol value 74
 program properties table 13
 RACF commands for authorizing CQS registration 33
 RACF commands to authorize connection to CQS structures 34
 RSRCSTRUCTURE= parameter 23
 SSN= parameter 14, 17
 starting CQS 27
 STEPLIB DD statement to concatenate
 IMS.SDFSRESL 77
 structure recovery data set 26
 system checkpoint data set 25

exit routines
 client 165
 Event 165
 user-supplied, CQS 47

F

file select utility 28
 formatting print utility 28
 functions of CQS 3

H

hardware requirements 6

I

IMS
 parameters, specifying 15
 IMS.ADFSMAC data set 77
 Initialization-Termination (Init-Term) user-supplied exit routine
 CQS 48
 interface
 authorization 70
 interrupt request block 165
 IRB 165

L

limiting log data 182
 lists, using with client request 75
 literals 74
 using 74
 log records
 control statements for printing 181
 copying 182
 DD statements for printing 181
 description 179
 JCL for printing 181
 limiting log data 182
 printing 181
 table 179
 types 179
 viewing format 179
 logging 6
 logical record length 16
 LOGR (system logger) policy 8
 LRECL 16

M

managing structure usage 37
 MAXBUFSIZE parameter 8
 message
 CQS0009W 29
 CQS0020I 30
 CQS0031A 30
 CQS0032A 30
 CQS0033A 32, 92

message (*continued*)
 CQS0034A 30, 41
 CQS0102E 32
 CQS0205E 37
 CQS0242E 59
 CQS0268I 22
 CQS0300I 45
 IXC585E 38
 IXC586I 38
 WTOR 30
 message queue 8
 MSGQ structures 8

O

OPTWORD1 parameter 74
 overflow
 mode 37
 processing 37
 threshold 37
 overflow processing 4

P

parameter
 OPTWORD1 74
 parameter lists
 abnormal termination 166
 Client Connection user exit 50
 Client Disconnect user exit 50
 Initialization user exit 49
 Queue Overflow user exit 52
 restart entry 166
 Structure Event exit routine
 checkpoint 171
 Deferred Resync Complete 169
 resync, CQS 169
 structure overflow 174
 structure rebuild 172
 structure rebuild lost UOWs 172
 structure status change 174
 Structure Event user exit 63
 checkpoint 64
 connect 63
 overflow 66
 Rebuild 65
 status change 67
 Structure Inform exit routine 176
 Structure Statistics user exit 54
 Termination user exit 49
 passing a value
 for register 73
 for symbol 73
 for symbol value 74
 passing an address
 for register 73
 for symbol 73
 passing an equate for symbol value 74
 performance tuning 78
 planning for CQS
 hardware requirements 6

- planning for CQS (*continued*)
 - software requirements 6
- policies
 - ARM 8
 - CFRM 8
 - defined 7
 - defining 8
 - failing 9
 - LOGR 8
 - SFM 8
- preventing structure full 37
- private queue types managed by CQS 4
- program properties table 13
- program, assembling 77

Q

- queue
 - cold 4
 - control 4
 - delete 4
 - lock 4
 - move 4
 - structure 3
 - type
 - client 4
 - private 4
 - private, managed by CQS 4
 - values 3
- Queue Overflow user-supplied exit routine
 - CQS 51
- queues
 - object on the cold queue 78
 - registering interest in 78

R

- RACF 33
 - commands for authorizing CQS registration 33
 - commands to authorize connection to CQS
 - structures 34
 - FACILITY class 33
- rebuild lost UOW entry, CQS 173
- rebuilding structures 39
- records restart 3
- recovery
 - functions 5
 - information 3
 - recovering CQS 39
 - structures 41
 - structures, for restart 29
- register
 - contents
 - Client Connection user exit 49, 50
 - Client Structure Event exit 168
 - Client Structure Inform exit 176
 - CQS Event exit 165, 166
 - Initialization-Termination user exit 48
 - Queue Overflow user exit 52
 - Structure Event user exit 63
 - Structure Statistics user exit 53, 54

- registers
 - client requests 72
 - using 72, 73
- registration, authorizing 33
- requests 69
 - assembling a program 77
 - authorization 70
 - coding 70
 - CQSBRWSE 80
 - CQSCHKPT 87
 - CQSCONN 90
 - CQSDEL 96
 - CQSDEREG 100
 - CQSDISC 101
 - CQSINFRM 106
 - CQSMOVE 110
 - CQSPUT 114
 - CQSQUERY 121
 - CQSREAD 130
 - CQSRECVR 135
 - CQSREG 140
 - CQSRSYNC 142
 - CQSSHUT 149
 - CQSUNLCK 150
 - CQSUPD 155
 - DSECTs, using 77
 - ECB, using 74
 - environmental requirements 71
 - example 159
 - introduction 69
 - lists, using 75
 - literals, coding 74
 - literals, using 74
 - parameters, coding 73
 - register, using 73
 - return and reason codes 75
 - sample 159
 - sequence of 70
 - symbol name, using 73
 - symbol value, using 73
- requirements
 - hardware 6
 - software 6
- resource
 - cleanup failure 32
 - structure
 - changes logged 28
 - recovery 41
 - structures 4
 - structures, and overflow processing 4
- restart
 - structure recovery 29
 - z/OS Automatic Restart Manager 31
- restarting CQS
 - cold start 31
 - description 30
 - structure initialization 28
 - warm start 30
- resync UOW entry, CQS 170
- return and reason codes
 - client requests 75

return and reason codes *(continued)*

CQSBRWSE request 86
 CQSCHKPT request 90
 CQSCONN request 95
 CQSDEL request 99
 CQSDEREG request 101
 CQSDISC request 105
 CQSINFRM request 110
 CQSMOVE request 113
 CQSPUT request 120
 CQSQUERY request 129
 CQSREAD request 134
 CQSRECVR request 139
 CQSREG request 141
 CQSRSYNC request 147
 CQSSHUT request 150
 CQSUNLCK request 154
 CQSUPD request 159

routines

client 165
 user-supplied, CQS 47

S

sequence of requests 70
 setting up a sysplex 7
 SFM (sysplex failure management) policy 8
 shutting down CQS 44, 78
 software requirements 6
 special events, handling.
 See events, handling
 starting CQS 12, 27
 statistic records
 CQS request 57
 data object 56
 queue name 57
 request 55
 structure checkpoint 60
 structure checkpoint entry 61
 structure process 55
 structure rebuild 58
 z/OS request 57
 STEPLIB DD statement to concatenate
 IMS.SDFSRESL 77
 structure
 alter 34
 authorizing connections to 33
 checkpoint, initiating 35
 cold start 29
 copy 42
 deleting 44
 duplexing 5, 42
 enabling 43
 EMHQ
 CQSSGxxx 20
 CQSSLxxx 18
 disabling 18, 20
 empty 28
 full, monitoring 38
 functions 4
 initialization 28

structure *(continued)*

overflow 4, 39
 function 37
 structure full monitoring 39
 pair 3
 rebuild 5
 initiating 40
 recovery 41
 recovery data set, example 26
 recovery for restart 29
 repopulation 41
 restarting CQS 28
 size 12, 34
 types 3
 warm start 28
 Structure Event user-supplied exit routine 62
 structure full
 managing 37
 Structure Statistics user-supplied exit routine 53
 structure usage
 managing 37
 structures
 resource 4
 supporting multiple clients 11
 symbol name, using 73
 symbol value, using 73
 sysplex
 setting up 7
 system checkpoint data set example 25
 system checkpoint, initiating 34
 system-managed rebuild 39

U

user exits (CQS).
 See CQS user-supplied exit routines
 user-managed rebuild 40
 user-supplied exit routines.
 See CQS user-supplied exit routines
 utilities
 DFSERA10 181
 file select 28
 formatting print 28
 IEBGENER 182
 IXGSEXIT 182
 printing log records 181
 z/OS logger subsystem exit 182
 UXPL_EXITPLP
 Client Connections exit 50
 Init-Term exit 49
 Queue Overflow exit 52
 Structure Statistics exit 54

W

warm start 30
 warm starting structures 28
 writing a CQS client 69
 WTOR 30

X

XCF 14

Z

z/OS

defining policies 7

program properties table 13

adding CQSINIT0 13

updating 13

z/OS Automatic Restart Manger 31



Program Number: 5655-J38

IBM Confidential
Printed in USA

ZES1-2339-02



Spine information:



IMS

Common Queue Server Guide and Reference

Version 9