IMS
15.3.0

System Programming APIs
(2024-08-30 edition)

Note

Before you use this information and the product it supports, read the information in “Notices” on page 535.

2024-08-30 edition.

This edition applies to IMS 15 (program number 5635-A06), IMS Database Value Unit Edition, V15.03.00 (program number 5655-DS5), IMS Transaction Manager Value Unit Edition, V15.03.00 (program number 5655-TM4), and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this information

These topics provide reference information for IMS system application programming interface (API) calls for IMS Common Queue Server (CQS); IMS Common Service Layer (CSL); IMS data propagation with IMS DataPropagator for z/OS®; IMS Database Resource Adapter (DRA); IMS Database Recovery Control (DBRC) API; IMS catalog API;IMS Repository Server (FRPBATCH); and VTAM® and SNA.

This information is available in IBM® Documentation.

Prerequisite knowledge

Before using this information, you should have knowledge of either IMS Database Manager (DB) or IMS Transaction Manager (TM). You should also understand basic z/OS and IMS concepts, your installation's IMS system, and have general knowledge of the tasks involved in project planning.

To learn about z/OS, see z/OS Basic Skills. For more resources, see IBM Z Education and Training.

To learn about IMS, see the IBM Press publication *An Introduction to IMS*, the resources listed for IBM Information Management System, and the variety of options available in IBM Training.

How new and changed information is identified

For most IMS library PDF publications, information that is added or changed after the PDF publication is first published is denoted by a character (revision marker) in the left margin. The *Program Directory* and *Licensed Program Specifications* do not include revision markers.

Revision markers follow these general conventions:

- Only technical changes are marked; style and grammatical changes are not marked.
- If part of an element, such as a paragraph, syntax diagram, list item, task step, or figure is changed, the entire element is marked with revision markers, even though only part of the element might have changed.
- If a topic is changed by more than 50%, the entire topic is marked with revision markers (so it might seem to be a new topic, even though it is not).

Revision markers do not necessarily indicate all the changes made to the information because deleted text and graphics cannot be marked with revision markers.

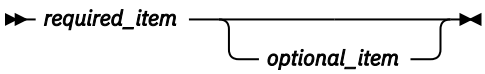
How to read syntax diagrams

The following rules apply to the syntax diagrams that are used in this information:

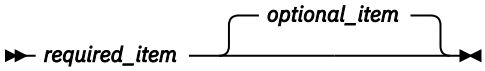
- Read the syntax diagrams from left to right, from top to bottom, following the path of the line. The following conventions are used:
  - The >>--- symbol indicates the beginning of a syntax diagram.
  - The ---> symbol indicates that the syntax diagram is continued on the next line.
  - The >--- symbol indicates that a syntax diagram is continued from the previous line.
  - The --->< symbol indicates the end of a syntax diagram.
- Required items appear on the horizontal line (the main path).

  ▶▶ *required_item* ⋈

- Optional items appear below the main path.

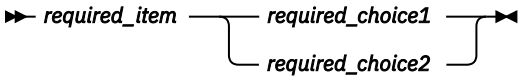


If an optional item appears above the main path, that item has no effect on the execution of the syntax element and is used only for readability.

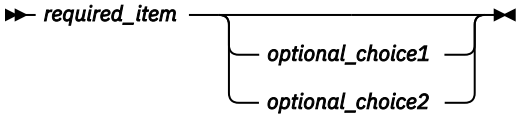


- If you can choose from two or more items, they appear vertically, in a stack.

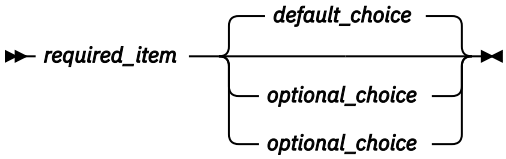
  If you *must* choose one of the items, one item of the stack appears on the main path.



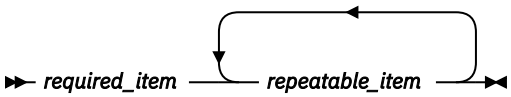
  If choosing one of the items is optional, the entire stack appears below the main path.



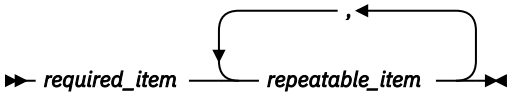
  If one of the items is the default, it appears above the main path, and the remaining choices are shown below.



- An arrow returning to the left, above the main line, indicates an item that can be repeated.

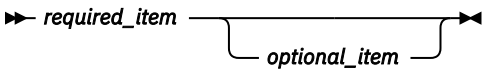


  If the repeat arrow contains a comma, you must separate repeated items with a comma.



  A repeat arrow above a stack indicates that you can repeat the items in the stack.
- 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.

  fragment-name



- In IMS, a b symbol indicates one blank position.

- Keywords, and their minimum abbreviations if applicable, appear in uppercase. They must be spelled exactly as shown. Variables appear in all lowercase italic letters (for example, *column-name*). They represent user-supplied names or values.
- Separate keywords and parameters by at least one space if no intervening punctuation is shown in the diagram.
- Enter punctuation marks, parentheses, arithmetic operators, and other symbols, exactly as shown in the diagram.
- Footnotes are shown by a number in parentheses, for example (1).

Accessibility features for IMS 15.3

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in z/OS products, including IMS 15.3. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers and screen magnifiers.
- Customization of display attributes such as color, contrast, and font size.

Keyboard navigation

You can access IMS 15.3 ISPF panel functions by using a keyboard or keyboard shortcut keys.

For information about navigating the IMS 15.3 ISPF panels using TSO/E or ISPF, refer to the *z/OS TSO/E Primer*, the *z/OS TSO/E User's Guide*, and the *z/OS 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.

Related accessibility information

Online documentation for IMS 15.3 is available in IBM Documentation.

IBM and accessibility

See the *IBM Human Ability and Accessibility Center* at www.ibm.com/able for more information about the commitment that IBM has to accessibility.

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 take one of the following actions:

- Submit a comment by using the DISQUS commenting feature at the bottom of any IBM Documentation topic.
- Send an email to imspubs@us.ibm.com. Be sure to include the book title.
- Click the **Contact Us** tab at the bottom of any IBM Documentation topic.

To help us respond quickly and accurately, please include as much information as you can about the content you are commenting on, where we can find it, and what your suggestions for improvement might be.

Part 1. Common Queue Server (CQS)

You can use this information to learn about writing a CQS client and CQS client requests.

Chapter 1. Writing a CQS client

Your CQS client communicates with CQS through requests. You must write one or more CQS clients in order to use CQS to manage resource and queue structures for your product or service.

There are various considerations that you must take into account when writing a Common Queue Server (CQS) client. The information in these topics is written primarily for the programmer who writes the client, but also for CQS administrators or system programmers who must become aware of some of the issues involved in designing and writing a CQS client.

This topic contains General-use Programming Interface information.

Related concepts

"CQS client requests" on page 13
CQS clients communicate with the CQS address space using a general-use interface consisting of a number of assembler macros, called CQS requests. CQS clients use these requests to communicate with the CQS and manipulate client data on shared coupling facility structures. You can use these requests to write or maintain a CQS client.

Summary of CQS client requests

CQS client requests enable a client to access CQS or shared queues on coupling facility list structures. Your primary tool for writing a CQS client is the set of client request macros that CQS provides. You can use these requests to enable a client to access CQS or the shared queues on coupling facility list structures.

The following list summarizes the CQS requests:

CQSBRWSE
Retrieves a copy of a data object from a queue

CQSCHKPT
Takes a checkpoint of internal tables or of all data objects on a structure

CQSCONN
Connects a client to one or more structures

CQSDEL
Deletes one or more data objects from a queue

CQSDEREG
De-register 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

Sequence of CQS requests issued by a client for queue structure

A client uses CQS requests to make use of CQS services and resources. Client requests for CQS services must be in a particular sequence, which is outlined in this table.

The client must issue certain requests to request CQS services, and some of the requests must be in a particular sequence; the sequence for CQS requests is shown in the following table. Other requests can be issued multiple times, in any order, based on the processing requirements of the client.

Table 1. Sequence for CQS requests

Order	Request	Use for 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 CQSBRWSE, 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 only 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.

Considerations for coding CQS requests

Various keywords, parameters, and variables are available for use with CQS requests. The interface that you select for the client's state determines the allowed environment for all subsequent CQS requests and all client exit routines driven by CQS.

The usage topic for each request describes the detail for each of the keywords, parameters, and variables for the CQS requests, but a few global usage considerations apply to all of the requests.

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 the client is authorized (in supervisor state with PSW key 0 to 7), CQS initializes the authorized interface environment. If the 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.

How CQS requests use registers

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. 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.

   Figure 1. Passing an address for register

   ```
            LA    5,TOKEN
            CQSREAD FUNC=READ,CQSTOKEN=(5),…
            ⋮
   TOKEN    DS    XL16
   ```

   Figure 2. Passing a value for register

   ```
            L     4,MYBUFLEN
            CQSREAD FUNC=READ,BUFSIZE=(4),…
            ⋮
   MYBUFLEN DC    F'00000024'
   ```

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.

Figure 3. Passing an address for symbol

```
CQSREAD FUNC=READ,CQSTOKEN=TOKENADR,…
⋮ TOKENADR DC    A(TOKEN) TOKEN    DS    XL16
```

Figure 4. Passing a value for symbol

```
CQSREAD FUNC=READ,BUFSIZE=MYBUFLEN,…
⋮ MYBUFLEN DC    F'00000024'
```

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.

   Figure 5. Passing a value for symbol value

```
CQSREAD FUNC=READ,CQSTOKEN=@(TOKEN),…
⋮ TOKEN    DC    XL16'0000A765B55CFF00'
```

   Figure 6. Passing an equate for symbol value

```
 CQSREAD FUNC=READ,BUFSIZE=@(MYBUFLEN),…
⋮ MYBUFLEN EQU   24
```

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....

Coding CQSREAD with the OPTWORD1 parameter

To code the same CQSREAD request using the OPTWORD1 parameter, use the example shown in the following example.

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

Event Control Blocks 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.

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.

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.

STEPLIB DD statement to concatenate IMS.SDFSRESL

To concatenate the IMS.SDFSRESL data set after your MYPROGS.SDFSRESL data set, code your STEPLIB DD statement as shown in the following example.

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

Related concepts

"CQS client requests" on page 13
CQS clients communicate with the CQS address space using a general-use interface consisting of a number of assembler macros, called CQS requests. CQS clients use these requests to communicate with the CQS and manipulate client data on shared coupling facility structures. You can use these requests to write or maintain a CQS client.

Related reference

z/OS: Initializing extended ECBs and ECB extensions

Environmental requirements for CQS

Environmental requirements depend on the CQS interface assigned to the client for CQS requests other than CQSREG and CQSDEREG requests.

The following table shows the environment for clients using the authorized CQS interface:

Table 2. 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

Table 2. Environment for CQS requests (excluding CQSREG and CQSDEREG) using the authorized interface (continued)

Environment	State
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

The following table shows the environment for clients using the non-authorized CQS interface:

Table 3. 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 the following table.

Table 4. 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

Table 4. Environment for CQSREG and CQSDEREG requests using the authorized interface (continued)

Environment aspect	State
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 the following table.

Table 5. 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

Return codes and reason codes for CQS requests

CQS return and reason codes indicate the success or failure of sending the request to the CQS address space and reflect the success or failure of the particular CQS request that is being made.

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 the following table. 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.

Attention:

For a complete list of CQS return codes and reason codes see the macro CQSRQSRR in the delivered SDFSMAC library. This MACRO contains high level codes and references other macros which contain function specific return codes and reason codes.

Table 6. Return and reason codes for errors detected by the CQS interface

Return code	Reason code	Meaning
X'00000008'	X'00000210'	The *cqstoken* is invalid.
X'00000008'	X'00000214'	The *connecttoken* 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.

Table 6. Return and reason codes for errors detected by the CQS interface (continued)

Return code	Reason code	Meaning
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.

CQS clients and handling special events

A CQS client must be able to either initiate or participate in many different types of events. You must be aware of what the CQS client can do in these events in order to handle them appropriately.

A CQS client must be able either to initiate or to participate in many different types of events. This topic 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 is notified of work on that queue.

Working with objects on the cold queue using CQS requests

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

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

```
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. A CQS client can initiate a structure checkpoint by issuing a CQSCHKPT FUNC=CHKPTSTR request.

Shut 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.

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 concepts

CQS administration (System Administration)

Related reference

“CQSQUERY request” on page 55
The CQSQUERY request retrieves information or status about one or more of the structures managed by CQS.

“CQSDEL request” on page 30
A CQSDEL request deletes one or more data objects from a queue structure or a resource structure.

“CQSINFRM request” on page 41
The CQSINFRM request registers or deregisters interest for one or more queues on a specific coupling facility structure.

Chapter 2. CQS client requests

CQS clients communicate with the CQS address space using a general-use interface consisting of a number of assembler macros, called CQS requests. CQS clients use these requests to communicate with the CQS and manipulate client data on shared coupling facility structures. You can use these requests to write or maintain a CQS client.

You do not need to use these requests 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.

This topic contains General-use Programming Interface information.

Example of using a CQS request: CQSREAD

The following example 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                                                     *
*                                                                     *
*********************************************************************
         STM   R14,R12,12(R13)     SAVE THE REGS
         LR    R12,R15             R12 = PROGRAM BASE REGISTER
         USING GETDOBJ,R12         GETDOBJ  CSECT
         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****
*        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*
* 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)
RDRPARM  DS    XL(CQSREAD_PARM_LEN) CQSREAD PARMLIST
*************************************************************************
*        LITERALS                                                       *
*************************************************************************

         LTORG
         CQSREAD FUNC=DSECT       CQSREAD DSECTS & EQUATES
         END   GETDOBJ
```

Related concepts

“Writing a CQS client” on page 3
Your CQS client communicates with CQS through requests. You must write one or more CQS clients in order to use CQS to manage resource and queue structures for your product or service.

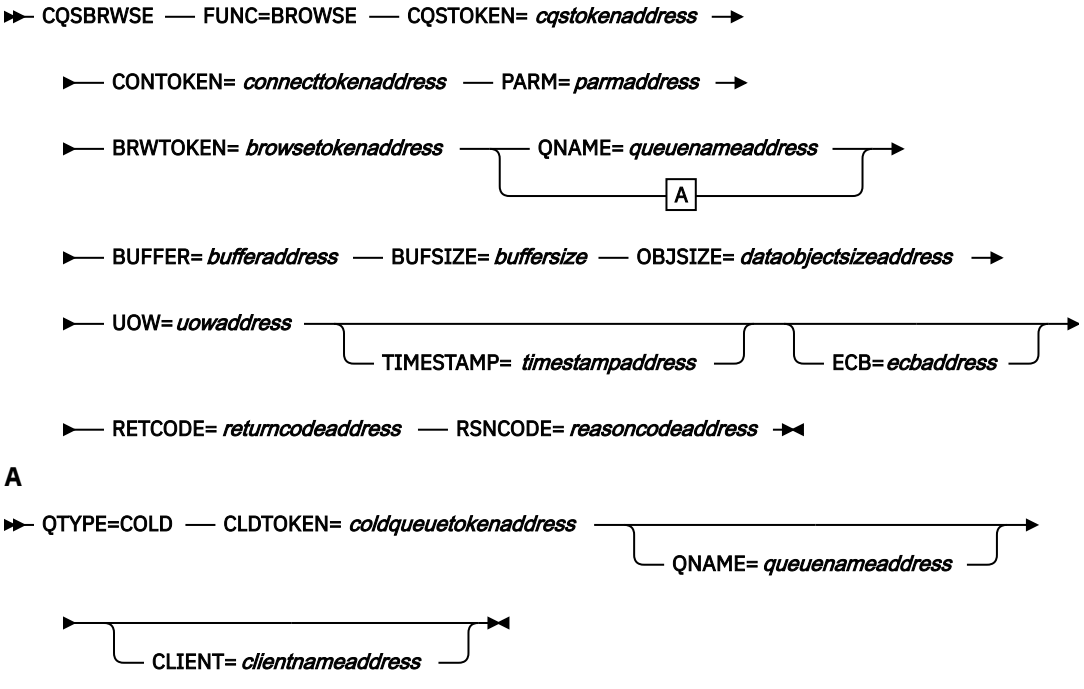
CQSBRWSE request

The CQSBRWSE request retrieves information from a specified queue or resource structure.

Format

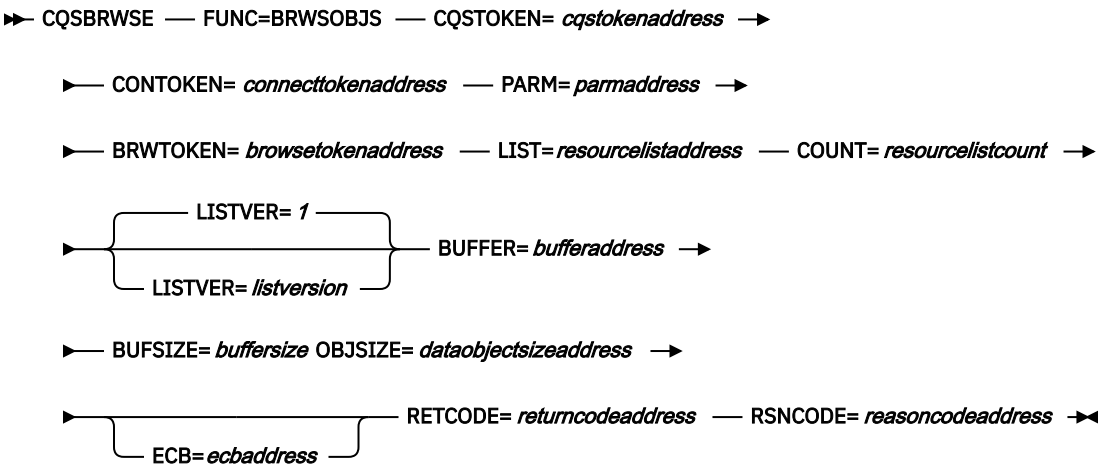
BROWSE function

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



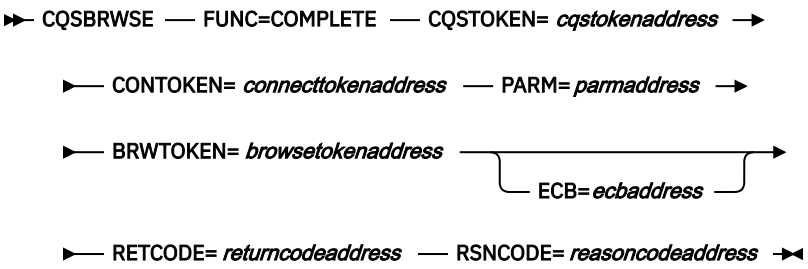
BRWSOBJS function

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



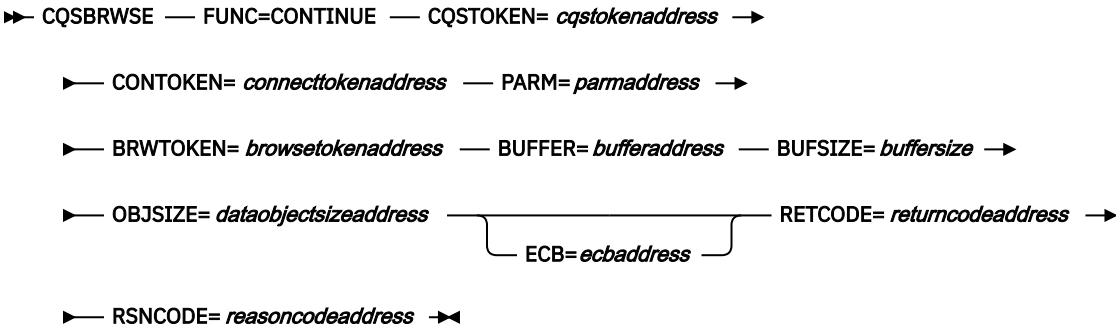
COMPLETE function

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



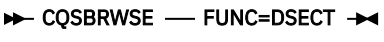
CONTINUE function of CQSBRWSE

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



DSECT function

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



Usage notes

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

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

When a CQSBRWSE FUNC=BROWSE request is issued and the buffer that is 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.

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

A CQSBRWSE FUNC=COMPLETE request indicates to CQS that the CQSBRWSE request that is associated with a browse token is complete. The browse token from the prior CQSBRWSE request is required. CQS invalidates the browse token and frees any data structures that are 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.

The CQSBRWSE FUNC=CONTINUE request is not supported for a resource structure because the CQSBRWSE FUNC=BRWSOBJS request does not return partial data.

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 one or more of the objects might be snapped by the first CQSBRWSE FUNC=BROWSE request and no longer be available because of another CQSREAD or CQSDEL request.
- 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 that no objects were 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. QSMOVE request, or overflow threshold processing. The CQSBRWSE FUNC=BROWSE simply skips objects that are no longer available.
- If the current position is lost because a browse table times out, a CQSBRWSE FUNC=CONTINUE request is rejected.

Parameters

BRWTOKEN=*browsetokenaddress*
Input and output parameter that specifies the address of the 16-byte browse token. The browse token maintains the cursor position of the data objects that are being browsed.

Set the browse token to zero on the initial CQSBRWSE request. Pass the browse token that is returned by CQS on a CQSBRWSE FUNC=BROWSE or FUNC=BRWSOBJS request as input on a subsequent CQSBRWSE=BROWSE, CQSBRWSE=CONTINUE, CQSBRWSE=COMPLETE, or CQSBRWSE=BRWSOBJS request.

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

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

BUFFER=*bufferaddress*
4-byte input parameter that specifies the address of a client buffer that holds information that is retrieved about one or more data objects.

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

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

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

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

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

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

Possible completion codes are:

X'00000000'
Request completed successfully.

X'00000020'
The *Resourceid* parameter 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 resource ID, 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 unit of work (UOW) on a CQSRECVR request to retrieve or delete objects on the cold queue.

CLIENT=*clientnameaddress*
4-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 the QTYPE=COLD parameter 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.

COUNT=*resourcelistcount*
4-byte input parameter that specifies the number of entries in the resource 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*
4-byte input parameter that specifies the address of the z/OS event control block (ECB) that is 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 that contains one or more entries. Each entry is a separate browse request. The client must initialize some fields in each entry before issuing the CQSBRWSE request. Other fields are returned by CQS when the request completes.

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 parameters:

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

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

reserved
3-byte reserved field.

owner
8-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
4-byte input parameter that specifies browse options. Possible options are:

X'80000000'
Return *data2* for the browsed data objects.

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

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

OBJSIZE=*dataobjectsizeaddress*
Output parameter that specifies the address of a 4-byte area to store 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=BRWSOBJS 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=BRWSOBJS 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*
4-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*
4-byte output parameter that specifies the address of a 16-byte queue name field.

For a CQSBRWSE request that specifies the QTYPE=COLD and CLDTOKEN parameters, the queue name field is an output field to contain the original client queue name for the data object being returned. This client queue name contained the data object before it was moved to the cold queue.

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

QTYPE=COLD
Input parameter that specifies the queue type from which the data object is to be retrieved. COLD Indicates that the data object is to be retrieved from the cold queue.

RETCODE=*returncodeaddress*
Output parameter that specifies the address of a 4-byte field to contain the CQSBRWSE return code.

If the return code in register 15 is a non-zero value, the values in the return and reason code fields are invalid, because the CQS interface detected an error and was unable to send the request to CQS.

RSNCODE=*reasoncodeaddress*
Output parameter that specifies the address of a 4-byte field to contain the CQSBRWSE reason code.

TIMESTAMP=*timestampaddress*
4-byte output parameter that specifies the address of an 8-byte field to contain the time stamp 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

The following table lists the return and reason code combinations that can be returned for CQSBRWSE requests. Use a CQSBRWSE FUNC=DSECT request to include equate (EQU) statements in your program for the return and reason codes.

Table 7. CQSBRWSE request return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000004'	X'00000120'	The buffer size (*buffersize*) is less than the data object size (*dataobjectsize*). Partial data is returned.
X'00000004'	X'00000124'	The buffer size (*buffersize*) 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 (*queuename*) specified.
X'00000004'	X'0000012C'	No partial data to return.
X'00000004'	X'00000138'	Request complete and the last data object is returned.
X'00000004'	X'0000013C'	No more data objects to return.
X'00000008'	X'00000210'	*cqstoken* is invalid.
X'00000008'	X'00000214'	*connecttoken* is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'0000021C'	*browsetoken* is invalid.
X'00000008'	X'00000220'	*queuename* is invalid.
X'00000008'	X'00000224'	*buffer* is invalid.
X'00000008'	X'00000228'	*buffersize* is invalid.
X'00000008'	X'0000022C'	*dataobjectsize* is invalid.
X'00000008'	X'00000230'	*uow* is invalid.
X'00000008'	X'00000234'	*browsetoken* is invalid.
X'00000008'	X'00000250'	*count* is invalid.
X'00000008'	X'00000254'	*listaddress* is invalid.
X'00000008'	X'0000027C'	A CQSBRWSE FUNC=BROWSE request is not allowed for a resource structure. A CQSBRWSE FUNC=CONTINUE request is not allowed for a resource structure. No partial data is returned from a resource structure.
X'00000008'	X'00000280'	A CQSBRWSE FUNC=BRWSOBJS request is not allowed for a queue structure.

Table 7. CQSBRWSE request return and reason codes (continued)

Return code	Reason code	Meaning
X'00000008'	X'00000284'	Parm *listversion* is invalid.
X'00000008'	X'00000288'	*listversion* is invalid.
X'00000010'	X'00000400'	A CQSRSYNC request is required for this structure.
X'00000010'	X'00000404'	Structure is inaccessible. Retry request later.
X'00000010'	X'00000408'	Current position lost. Reissue a CQSBRWSE request.
X'00000010'	X'00000430'	No CQS address space.
X'00000014'	X'00000500'	CQS internal error.

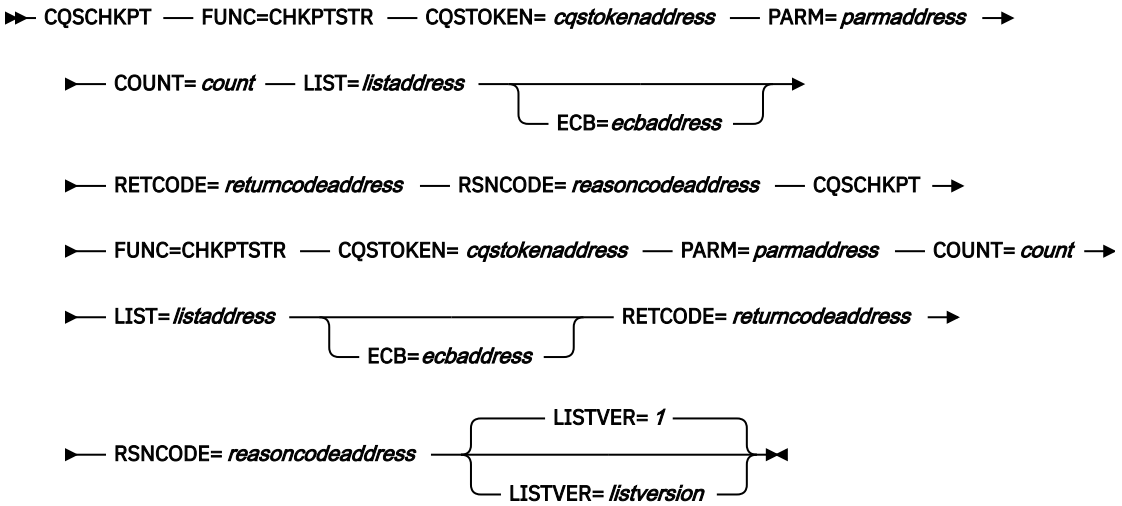
CQSCHKPT request

Use the CQSCHKPT request to initiate either a CQS system checkpoint or a structure checkpoint.

Format for CQSCHKPT

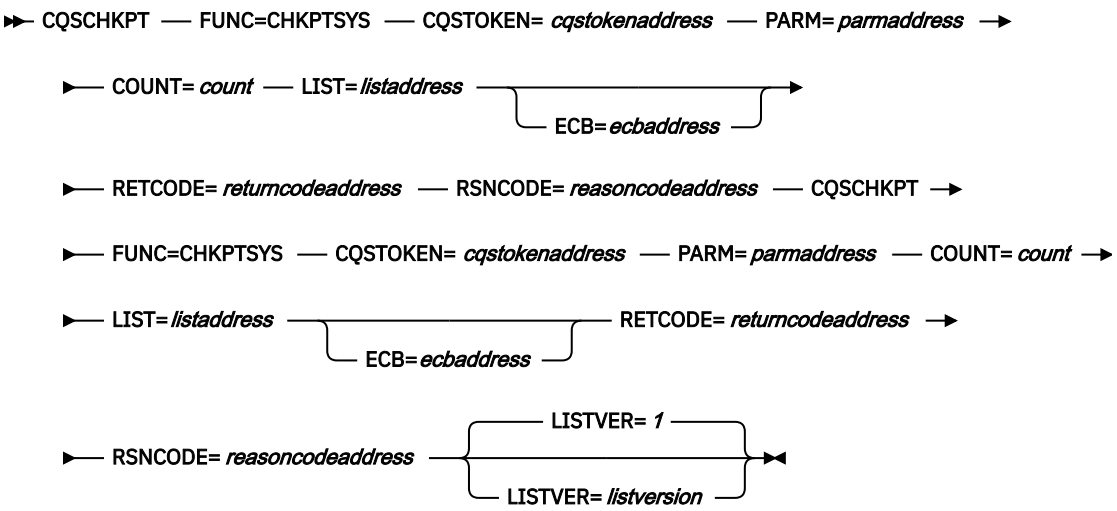
CHKPTSTR function of CQSCHKPT

You 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.



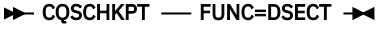
CHKPTSYS function of CQSCHKPT

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

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 descriptions

COUNT=*count*
4-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*
4-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*
4-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 parameters:

connecttoken
16-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
4-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.

X'00000024'
z/OS logger write error, checkpoint was not taken.

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*
4-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 statement 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

The following table 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 8. CQSCHKPT return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000008'	X'00000210'	*cqstoken* is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000250'	*count* is invalid.
X'00000008'	X'00000254'	*listaddress* is invalid.

Table 8. CQSCHKPT return and reason codes (continued)

Return code	Reason code	Meaning
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 *compcode* for individual errors.
X'0000000C'	X'00000304'	Request failed for all list entries. See *compcode* 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.

Related concepts

Using CQS system checkpoint (System Administration)

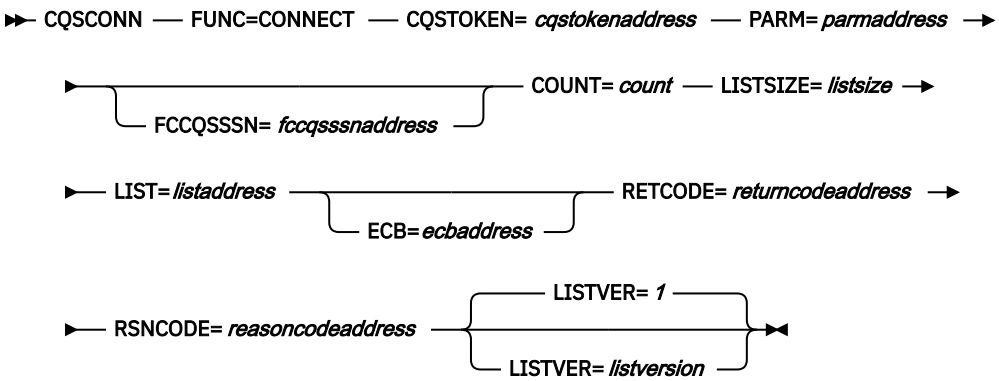
CQSCONN request

The CQSCONN request connects a client to one or more coupling facility structures.

Format for CQSCONN

CONNECT function of CQSCONN

You use the CONNECT function of a CQSCONN request to connect to one or more coupling facility structures. The coupling facility structures can be queue structures or resource structures.



DSECT function of CQSCONN

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

Usage of CQSCONN

The CQSCONN request connects a client to one or more coupling facility structures. The client specifies a connect list containing one or more list entries, for which each entry is a separate connect request. If the connection to a structure is successful, a connect token is returned to the client, representing the connection to the structure. The client must specify this token on all subsequent CQS requests for

that structure. A maximum of 32 clients can use a CQS address space to connect to a coupling facility structure.

Restriction: The CQSCONN request is not logged for resource structures and does not support the FCCQSSSN keyword. The CQSCONN request does not support the following connect list parameters for a resource structure:

- *structureattributes*
- *overflowstructurename*
- *structureinformexit*
- *structureinformparm*
- *qtypecnt*
- *qtypelist*

A CQSCONN FUNC=CONNECT request must be issued after a CQSREG FUNC=REGISTER request and before any other CQS requests. Also, after a CQS abnormal termination and restart, and after the client has reregistered with CQS, a CQSCONN FUNC=CONNECT request is required before the client can issue any other CQS requests.

Parameter Description:

COUNT=*count*
Four-byte input parameter that specifies the number of list entries in the connect list.

CQSTOKEN=*cqstokenaddress*
Input parameter that specifies the address of the 16-byte CQS registration token 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.

Note: All fields in the CQSCONNL DSECT currently documented as "Not Used" must be set to zero by the caller of CQSCONN.

Each list entry contains the following parameters:

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

X'00000000'
Client connection successful. A connect token is returned to the client.

X'00000004'
The client is already connected to the structure through this CQS. A connect token is returned to the client.

X'00000008'
structurename is invalid.

X'0000000C'
The Structure Event exit routine address was not specified.

X'00000010'
The client is already connected to the structure through another CQS. A client can only be connected to a given structure through one CQS. The client is not connected to the structure through this CQS. This does not affect the status of a client connection with another CQS.

X'00000014'
CQS internal error.

X'00000018'
The client specified the FCCQSSSN= parameter to connect to the structure to take over work for a failed client. CQS could not find a valid system-checkpoint log token for the CQS that was connected to the failed client. CQS issued message CQS0033A, to which the operator replied REJECT.

X'0000001C'
The user ID of the client address space is not authorized to connect to the structure.

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 *structureattributes* remains in effect for the life of the structure, and cannot be changed.

When set to 0, indicates that client requests to write and retrieve data objects from the structure do not wait for a rebuild to complete.

When set to 1, indicates that client requests to write and retrieve data objects from the structure must wait for a rebuild to complete.

X’40’
Output flag returned by CQS. For queue structures only, this flag indicates whether the structure is a non-recoverable structure (whether RECOVERABLE=NO was specified in the CQSSGxxx PROCLIB member for the structure). This flag is set to 1 if the structure is a non-recoverable structure; otherwise, it is set to 0.

This flag is not applicable to a resource structure.

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.

When using version 1 of the CQSCONN parameter list (the default), build the queue type list starting at label CNLQTYPL in the CQSCONNL DSECT, which maps the list entry. When using version 16 of the CQSCONN parameter list, build the queue type list starting at label CNLQTYPL_V16.

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.

logstreamname
Twenty-six-byte output parameter field to receive the name of the z/OS log stream associated with the CQS structure. This field is set to all blanks for non-recoverable queue structures and for resource structures.

This field is present only for CQSCONN lists at version 16 or later.

logstreamstructurename
Sixteen-byte output parameter field to receive the name of the CF structure associated with the z/OS log stream that is associated with the CQS structure. This field is set to all blanks for non-recoverable queue structures, resource structures, and structures with DASD-only z/OS log streams.

This field is present only for CQSCONN lists at version 16 or later.

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 the parameter list version. Use the DSECT function of the CQSCONN request to include equate (EQU) statements in your program for the CQSCONN list versions and lengths. The following parameter list versions are supported:

1
EQU symbol is CNL_LVER1. This is the default parameter list version. This version of the parameter list includes all fields documented under the LIST= parameter except for those that are specifically noted as being present only in a higher list version. The minimum length of a version 1 parameter list entry is CNL_MINLNV1 bytes. Queue type entries, if present, begin at label CNLQTYPL in the CQSCONNL DSECT, mapping the list entry.

16
EQU symbol is CNL_LVER16. A version 16 parameter list contains additional output fields beyond the fields present in a version 1 parameter list. These additional fields are documented under the LIST= parameter and are returned only when a version 16 format parameter list is passed. The minimum length of a version 16 parameter list entry is CNL_MINLNV16 bytes. Queue type entries, if present, begin at label CNLQTYPL_V16 in the CQSCONNL DSECT, mapping the list entry.

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

The following table 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 9. 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'	*cqstoken* is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000250'	*count* is invalid.
X'00000008'	X'00000254'	*listaddress* is invalid.
X'00000008'	X'00000258'	*listsize* 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 *compcode* for individual errors.
X'0000000C'	X'00000304'	Request failed for all list entries. See *compcode* for individual errors.
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.

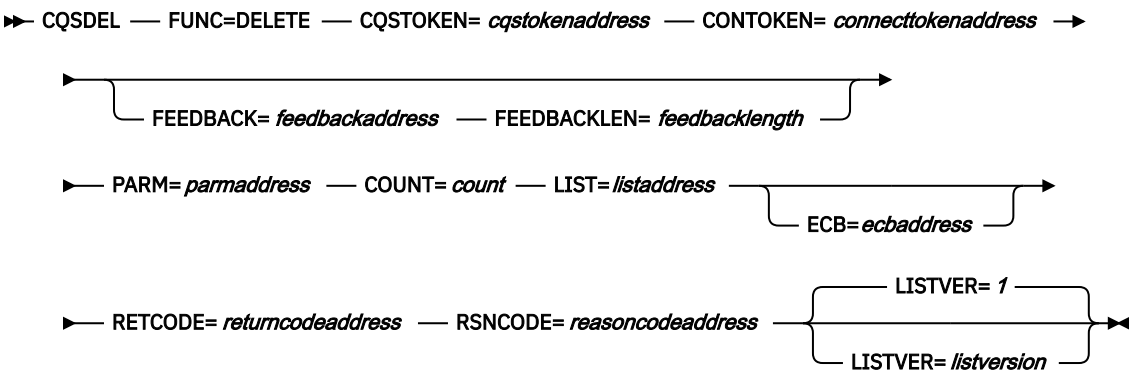
CQSDEL request

A CQSDEL request deletes one or more data objects from a queue structure or a resource structure.

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 that contains one or more list entries, for which each list entry is a separate delete request (either by lock token, by queue name, by queue name and UOW, by resource ID, or by resource type and owner). Each list entry is processed separately and receives its own completion code.

Parameter description:

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

COUNT=*count*
A 4-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*
A 4-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.

FEEDBACK=*feedbackaddress*
A 4-byte input parameter that specifies the address of a feedback area to receive structure utilization feedback about the CQS structure. This information includes the number of entries and elements that are allocated in the structure and the number currently in use. If the structure is in overflow mode, information about both the primary and overflow structure is returned. This parameter is optional for FUNC=DELETE; it is invalid for other CQSDEL functions. FEEDBACK= is required when FEEDBACKLEN= is specified.

The utilization data returned is guaranteed only to be current for the structures from which the data objects that were specified in the list entries by the LIST= parameter were actually deleted. Data for structures (overflow or primary) that were not accessed is as of the last time CQS put or deleted an object to that other structure, and might be obsolete if that last access was far in the past. This is because CQS receives the structure usage data back from the structure when it issues a write request or a delete request to the structure. CQS saves this information about both primary and overflow structures. However, if CQS does not receive requests to access one of the structures for a period of time, that structure's saved data can become out of date if other CQSs in the IMSplex are updating it.

You should consider the data returned in the FEEDBACK area as an approximation of the structures' usage, and not necessarily a precise measurement.

The format of the feedback area is described by mapping macro CQSSFBA – the CQS Structure Feedback Area. This area contains the Product-Sensitive Programming Interface (PSPI). For details about the feedback area, see the CQSSFBA macro shipped with IMS. The fields that are related to the overflow structure are meaningful only when the overflow structure is allocated. They are zero when there is no overflow structure. If the CQSDEL request receives a nonzero return code, the feedback area data might not be returned, depending on how far CQSDEL processing progressed before the failure. For a nonzero CQSDEL return code, either the feedback area is unchanged (no data returned) or it contains the current structure usage data (data returned).

FEEDBACKLEN=*feedbacklength*
A 4-byte input parameter that specifies the size of the feedback area specified by the FEEDBACK parameter. The FEEDBACKLEN parameter is optional for FUNC=DELETE. FEEDBACKLEN= is required when FEEDBACK= is specified.

SFBA_HDR_LN, the EQU symbol for the feedback area header section length, is defined in the CQSSFBA macro to be the maximum length of the area required for a given CQS version. If you pass a length less than the minimum value, then CQS returns no feedback data.

The feedback area total area length EQU symbol SFBA_LN is defined in the CQSSFBA macro to be the maximum length of the area required for a given CQS version. This EQU value might change in the future if new fields are added. If you cannot tolerate the length of your feedback area changing, you should define your own length value based on the `offset+length` of the last field in the CQSSFBA DSECT mapping that your program accesses.

You can pass a length less than the maximum value. CQS will truncate the feedback data to fit into your passed area. You can also pass a length greater than the maximum value. CQS will copy back the entire mapped feedback area; any excess storage beyond the area's length is unpredictable on return from CQS.

The feedback area length field SFBA_LENGTH in the header section is an output parameter. It is assigned the maximum length of the feedback area at run time and can be used to obtain the complete feedback data.

If you pass an EQU symbol as the length of the feedback area, ensure that you use the FEEDBACKLEN=@(symbol) notation. Use FEEDBACKLEN=symbol when you pass a symbol that is a label on a word of storage that contains the feedback area length.

LIST=*listaddress*
A 4-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 parameters:

deletetype
One-byte input parameter field that contains the delete type. This is a required parameter. *deletetype* can have one of the following values:

1
Delete by lock token.

2
Delete by queue name.

3
Delete by queue name and unit of work.

4
Delete by resource ID and version.

5
Delete by resource type with the specified owner.

Recommendation: For better performance, use delete type 1 or delete type 2 because they are more efficient than delete type 3.

deleteqpos
One-byte input parameter field that specifies either that all data objects are to be deleted or the position on the queue of data objects to be deleted. This parameter is only used for delete type 2. *deleteqpos* can have one of the following values:

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
A 2-byte reserved field.

objdelcnt
A 4-byte output parameter field to receive the number of data objects deleted.

compcode
A 4-byte output field to receive the completion code from the request. Possible completion codes are:

X'00000000'
Request completed successfully.

X'00000004'
Invalid *deleteqpos* (Delete type 2).

X'00000008'
Invalid *deletetype*.

X'0000000C'
Invalid *locktoken* (Delete type 1).

X'00000010'
Invalid *queuename* (Delete type 2 or type 3).

X'00000014'
Invalid *uow* (Delete type 3).

X'0000001C'
Structure is inaccessible. Try the request again later.

X'00000020'
CQS internal error.

X'00000024'
Data object not found on queue (Delete type 2) or on *queuename* for UOW (Delete type 3), or on resource structure (Delete type 4). It is up to the client to determine whether this case should be treated as an error or not.

X'00000028'
Delete type 1, 2, or 3 is invalid for a resource structure.

X'00000032'
Delete type 4 or 5 is invalid for a queue structure.

X'00000036'
Resourceid is invalid. The name type must be a decimal number between 1 - 255.

X'00000040'
Version is invalid. The version must be a number greater than zero.

X'00000044'
Version does not match that of an existing resource.

X'00000048'
Resourcetype is invalid. The resource type must be a decimal number between 1 - 255.

locktoken
A 16-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
A 16-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
A 32-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
A 12-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 *resourceytpe* fields are mutually exclusive.

version
An 8-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
A 1-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
A 3-byte reserved field.

owner
An 8-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*
A 4-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

The following table 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 10. CQSDEL return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000008'	X'00000210'	*cqstoken* is invalid.
X'00000008'	X'00000214'	*connecttoken* is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000250'	*count* is invalid.
X'00000008'	X'00000254'	*listaddress* is invalid.
X'00000008'	X'00000284'	Parameter list 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 *compcode* for individual errors.
X'0000000C'	X'00000304'	Request failed for all list entries. See *compcode* 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.

Related concepts
Monitoring shared message queue usage with the Queue Space Notification exit routine (DFSQSSP0) (System Administration)
Related reference
Queue Space Notification exit routine (DFSQSPC0/DFSQSSP0) (Exit Routines)

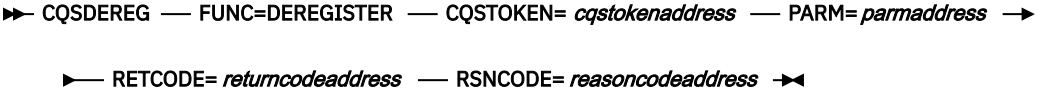
CQSDEREG request

The CQSDEREG request deregisters a client from CQS and invalidates the CQSTOKEN.

Format for CQSDEREG

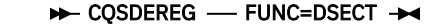
DEREGISTER function of CQSDEREG

A CQSDEL request deletes one or more data objects from a queue structure or a resource structure.



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.



Usage of CQSDEREG

The CQSDEREG request deregister 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

The following table lists the return and reason code combinations that can be returned for CQSDEREG requests.

Table 11. 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 *cqstoken* is now invalid.
X'00000004'	X'00000108'	Unable to delete z/OS Resource Manager routine. The *cqstoken* is now invalid.
X'00000008'	X'00000210'	*cqstoken* 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 *cqstoken* is now invalid.
X'00000014'	X'00000504'	Storage allocation error for work area.
X'00000014'	X'00000518'	CQS internal error (unable to create ESTAE).

Table 11. CQSDEREG return and reason codes (continued)

Return code	Reason code	Meaning
X'00000014'	X'0000053C'	Unable to load CQS deregistration module CQSREG10.

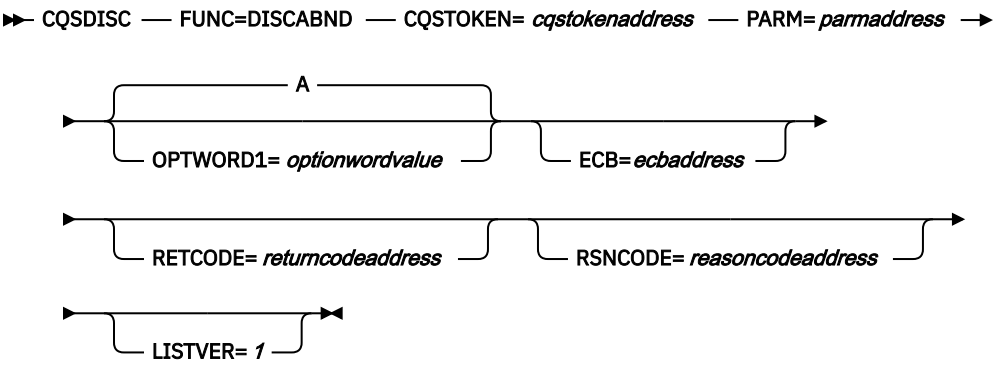
CQSDISC request

The CQSDISC request allows a client to disconnect from one or more coupling facility structures.

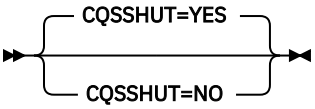
Format for CQSDISC

DISCABND function of CQSDISC

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

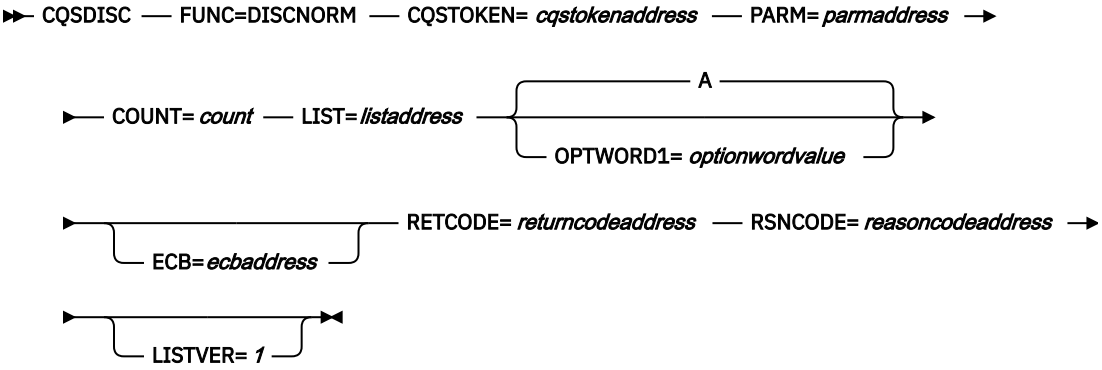


A

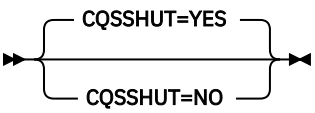


DISCNORM function of CQSDISC

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.

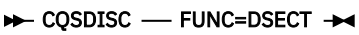


A



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 the request 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

The following table 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 12. 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'.
X'00000008'	X'00000210'	*cqstoken* is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000250'	*count* is invalid.
X'00000008'	X'00000254'	*listaddress* 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 *compcode* for individual errors.
X'0000000C'	X'00000304'	Request failed for all list entries. See *compcode* for individual errors.
X'00000010'	X'00000430'	No CQS address space.

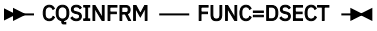
CQSINFRM request

The CQSINFRM request registers or deregisters interest for one or more queues on a specific coupling facility structure.

Format for CQSINFRM

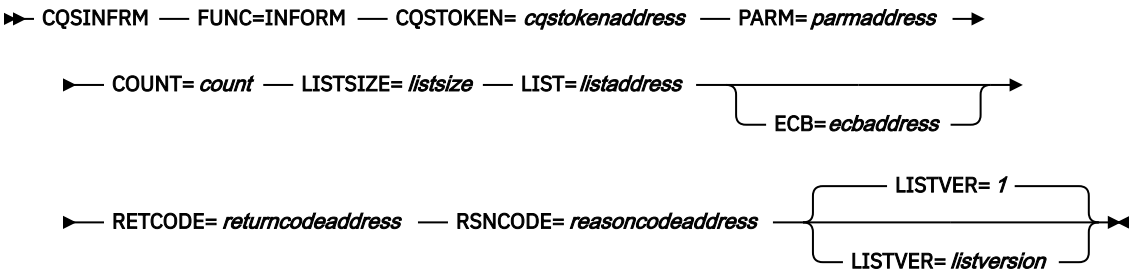
DSECT function of CQSINFRM

You 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.



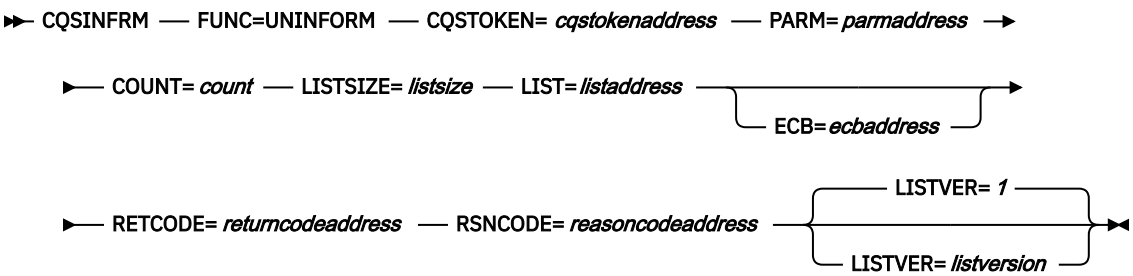
INFORM function of CQSINFRM

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.



UNINFORM function of CQSINFRM

Use the UNINFORM function of a CQSINFRM request to deregister a client's interest in one or more queues on a specific coupling facility structure it previously registered interest for.



Usage of CQSINFRM

A client uses a CQSINFRM request to register or deregister interest for one or more queues on a specific coupling facility structure. When a queue goes from empty to non-empty, CQS notifies all clients that registered interest for the queue of the change in status by scheduling the Structure Inform Client exit routine.

Restriction: The CQSINFRM request is not supported for resource structures.

The client can issue CQSREAD or CQSBRWSE requests to retrieve data from a queue. A client can make data objects available on a queue using CQSPUT, CQSMOVE, or CQSUNLCK requests.

A client that has registered interest in a queue is only notified when the queue goes from empty to non-empty, or if a data object is available on the queue when the CQSINFRM request is issued. The client does not receive notification when additional data objects are placed on a non-empty queue.

After a client deregisters interest in a queue, it is no longer notified when one of the queues goes from empty to non-empty. Because client notifications occur asynchronously with CQSINFRM requests, the client should expect to be notified about new data objects that arrive between the time the client issues the CQSINFRM FUNC=UNINFORM request and the time CQS processes the request.

Parameter Description:

COUNT=*count*
Four-byte input parameter that specifies the number of structure list entries in the structure list.

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

ECB=*ecbaddress*
Four-byte input parameter that specifies the address of the 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 parameters:

connecttoken
Sixteen-byte input parameter that specifies the connect token that uniquely identifies the client's connection to CQS and a specific coupling facility structure. The connect token is returned by the CQSCONN request. This parameter is required.

structurecompletioncode
Four-byte output field to receive the completion code for the CQSINFRM request for the structure. Possible structure completion codes are:

X'00000000'
Request completed successfully.

X'00000004'
Request completed successfully for all queues. At least one queue has work on it. See the queue completion code to determine which queues have work on them.

X'00000010'
connecttoken is invalid.

X'00000014'
queuelistcount is invalid.

X'00000018'
Inform exit routine does not exist. The Structure Inform exit routine was not specified on CQSCONN request for structure.

X'00000020'
Request completed successfully for at least one, but not all queues in *queuelist*. See *queuecompletioncode* for individual errors.

X'00000024'
Request failed for all queues in *queuelist*. See *queuecompletioncode* for individual errors or successes.

X'00000030'
A 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

The following table 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 13. 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'	*cqstoken* is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000250'	*count* is invalid.
X'00000008'	X'00000254'	*listaddress* is invalid.
X'00000008'	X'00000258'	*listsize* 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 *structurecompletioncode* for individual errors or successes.
X'0000000C'	X'00000304'	Request failed for all list entries. See *structurecompletioncode* for individual errors.
X'00000010'	X'00000430'	No CQS address space.

Related reference
CQS Client Structure Inform exit routine (Exit Routines)

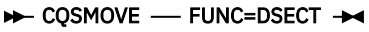
CQSMOVE request

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.

Format for CQSMOVE

DSECT function of CQSMOVE

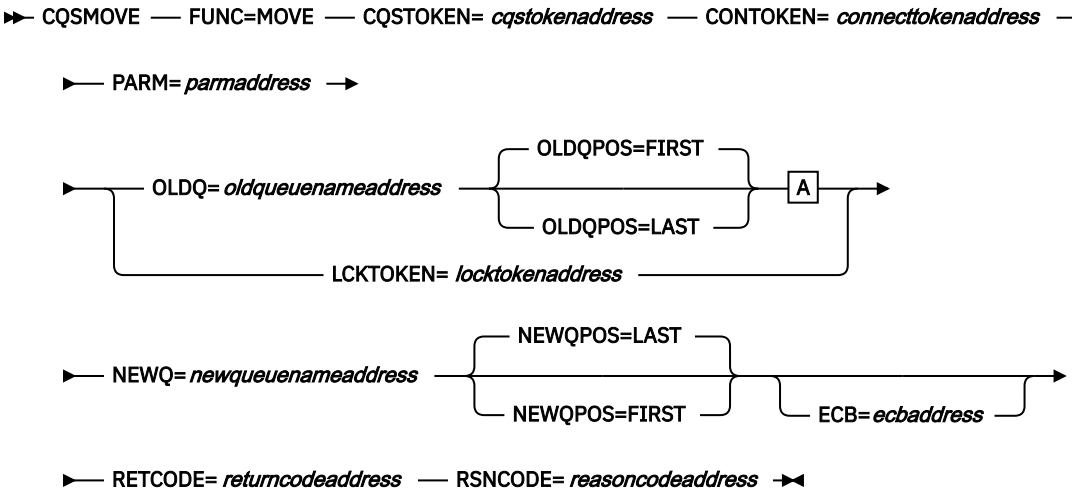
You use the DSECT function of a CQSMOVE request to include equate (EQU) statements in your program for the CQSMOVE parameter list length, CQSMOVE return and reason codes, and literals that can be used to build the OPTWORD1 parameter.



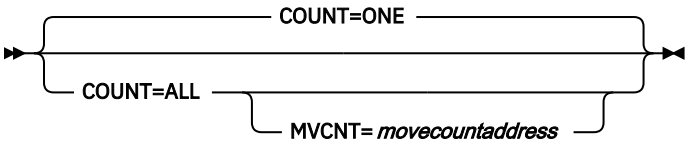
MOVE function of CQSMOVE

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

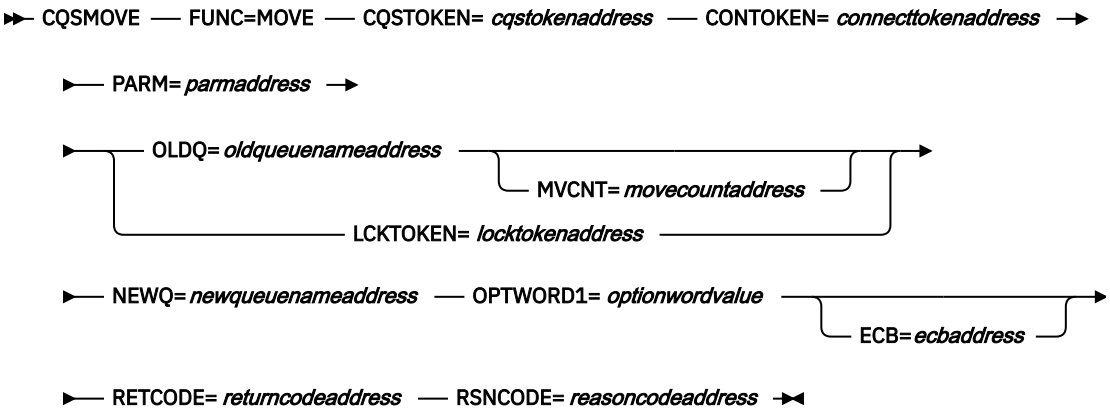


A



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=*newqueuenameaddress*
Input parameter that specifies the address of the 16-byte name of the new queue to which the data object is to be moved.

NEWQPOS=FIRST | LAST
Input parameter that specifies the position on the new queue to which data objects are moved, either first or last.

The NEWQPOS parameter cannot be used when the OPTWORD1 parameter is specified. If the OPTWORD1 parameter is specified instead of NEWQPOS, you can use the following equate (EQU) statements to generate the value for the OPTWORD1 parameter:

```
CQSMOVE_NEWQFEQUX   NEWQPOS=FIRST
CQSMOVE_NEWQLEQUX    NEWQPOS=LAST
```

OLDQ=*oldqueuenameaddress*
Input parameter that specifies the address of the 16-byte name of the old queue from which the data object is to be moved.

OLDQPOS=FIRST | LAST
Input parameter that specifies the position on the old queue from which data objects are to be moved, either first or last.

The OLDQPOS parameter cannot be used when the OPTWORD1 parameter is specified. If the OPTWORD1 parameter is specified instead of OLDQPOS, you can use the following equate (EQU) statements to generate the value for the OPTWORD1 parameter:

```
CQSMOVE_OLDQFEQUX   OLDQPOS=FIRST
CQSMOVE_OLDQLEQUX    OLDQPOS=LAST
```

OPTWORD1=*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

The following table 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 14. 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'	*cqstoken* is invalid.
X'00000008'	X'00000214'	*connecttoken* is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'0000021C'	*locktoken* 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.

Table 14. CQSMOVE return and reason codes (continued)

Return code	Reason code	Meaning
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 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.
X'00000014'	X'00000504'	z/OS logger write error, data objects were not moved.

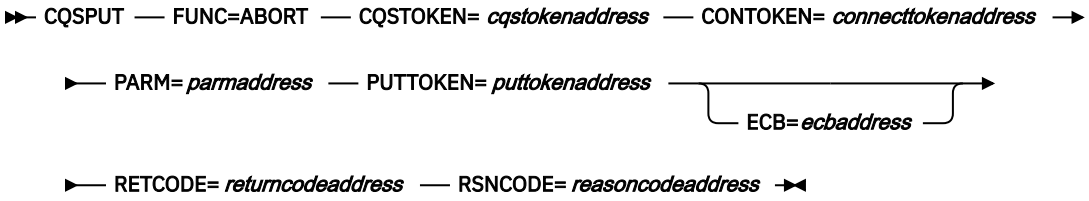
CQSPUT request

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.

Format for CQSPUT

ABORT function of CQSPUT

Use the ABORT function of a CQSPUT request to remove all uncommitted data objects from the queues that are associated with a recoverable unit of work.

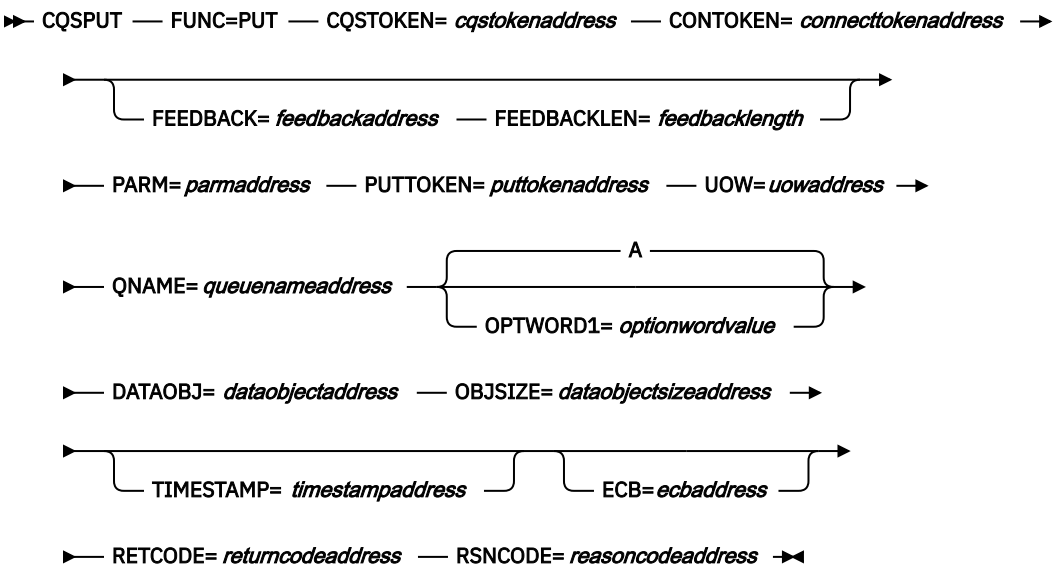


DSECT function of CQSPUT

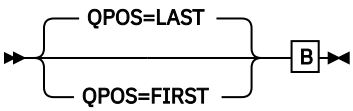
Use the DSECT function of a CQSPUT request to include equate (EQU) statements in your program for the CQSPUT parameter list length, CQSPUT return and reason codes, and literals that can be used to build the OPTWORD1 parameter.

PUT function of CQSPUT

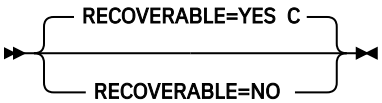
Use the PUT function of a CQSPUT request to place a data object on a queue.



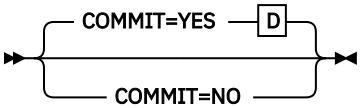
A



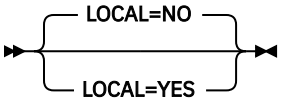
B



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. The following table shows the actions taken for each case.

When a data object is put on a queue, a time stamp is stored with the data object. The source of the time stamp 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 time stamp representing the current time is generated and stored with the data object. The time stamp 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 15. 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 nonrecoverable 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 nonrecoverable 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*
A 4-byte input parameter that specifies the address of the client data object to be placed on the specified queue.

ECB=*ecbaddress*
A 4-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.

FEEDBACK=*feedbackaddress*
A 4-byte input parameter that specifies the address of a feedback area to receive structure utilization feedback about the CQS structure. This information includes the number of entries and elements that are allocated in the structure and the number currently in use. If the structure is in overflow mode, information about both the primary and overflow structure is returned. This parameter is optional for FUNC=PUT; it is invalid for other CQSPUT functions. FEEDBACK= is required when FEEDBACKLEN= is specified.

The utilization data returned is guaranteed only to be current for the structures to which the data object passed on the DATAOBJ= parameter was actually written. Data for the structures (overflow or primary) that were not accessed is as of the last time CQS put or deleted an object to that other

structure, and might be obsolete if that last access was far in the past. This is because CQS receives the structure usage data back from the structure when it issues a write request or a delete request to the structure. CQS saves this information about both primary and overflow structures. However, if CQS does not receive requests to access one of the structures for a period of time, that structure's saved data can become out of date if other CQSs in the IMSplex are updating it. You should consider the data returned in the FEEDBACK area as an approximation of the structures' usage, and not necessarily a precise measurement.

The format of the feedback area is described by mapping macro CQSSFBA – the CQS Structure Feedback Area. This area contains the Product-Sensitive Programming Interface (PSPI). For details about the feedback area, see the CQSSFBA macro shipped with IMS. The fields that are related to the overflow structure are meaningful only when the overflow structure is allocated. They are zero when there is no overflow structure. If the CQSPUT request receives a nonzero return code, the feedback area data might not be returned, depending on how far CQSPUT processing progressed before the failure. For a nonzero CQSPUT return code, either the feedback area is unchanged (no data returned) or it contains the current structure usage data (data returned).

FEEDBACKLEN=*feedbacklength*

A 4-byte input parameter that specifies the size of the feedback area specified by the FEEDBACK parameter. The FEEDBACKLEN parameter is optional for FUNC=PUT; it is invalid for other CQSPUT functions. FEEDBACKLEN= is required when the FEEDBACK= parameter is specified.

SFBA_HDR_LN, the EQU symbol for the feedback area header section length, is defined in the CQSSFBA macro to be the minimum length of the area required for a given CQS version. If you pass a length less than the minimum value, CQS returns no feedback data.

SFBA_LN, the EQU symbol for the feedback area total area length, is defined in the CQSSFBA macro to be the maximum length of the area required for a given CQS version. This EQU value might change in the future if new fields are added. If you cannot tolerate the length of your feedback area changing, you should define your own length value based on the `offset+length` of the last field in the CQSSFBA DSECT mapping that your program accesses.

You can pass a length less than the maximum value. CQS will truncate the feedback data to fit into your passed area. You can also pass a length greater than the maximum value. CQS will copy back the entire mapped feedback area; any excess storage beyond the area's length is unpredictable on return from CQS.

The feedback area length field SFBA_LENGTH in the header section is an output parameter. It is assigned the maximum length of the feedback area at run time and can be used to obtain the complete feedback data.

If you pass an EQU symbol as the length of the feedback area, ensure that you use the FEEDBACKLEN=@ (symbol) notation. Use FEEDBACKLEN=*symbol* when you pass a symbol that is a label on a word of storage that contains the feedback area length.

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 CQSs.

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 impact 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*
A 4-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*
A 4-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*
A 4-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 because 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 that 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 that 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*
A 4-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

The following table 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 16. CQSPUT return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000008'	X'00000210'	*cqstoken* is invalid.
X'00000008'	X'00000214'	*connecttoken* is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'0000021C'	*puttoken* is invalid.
X'00000008'	X'00000220'	*queuename* is invalid.
X'00000008'	X'00000224'	*dataobject* is invalid.
X'00000008'	X'00000228'	*dataobjectsize* is invalid.
X'00000008'	X'00000230'	*uow* 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 16. 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 nonrecoverable. 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'	Parameter list version is invalid.
X'00000010'	X'00000400'	A CQSRSYNC is required for this structure.
X'00000010'	X'00000404'	Structure inaccessible. Try the request again later.
X'00000010'	X'00000414'	Queue for *queuename* 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.
X'00000014'	X'00000504'	z/OS logger write error, data objects were not placed on queues.

Related concepts

Monitoring shared message queue usage with the Queue Space Notification exit routine (DFSQSSP0) (System Administration)

Related reference

Queue Space Notification exit routine (DFSQSPC0/DFSQSSP0) (Exit Routines)

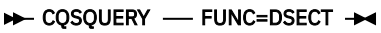
CQSQUERY request

The CQSQUERY request retrieves information or status about one or more of the structures managed by CQS.

Format for CQSQUERY

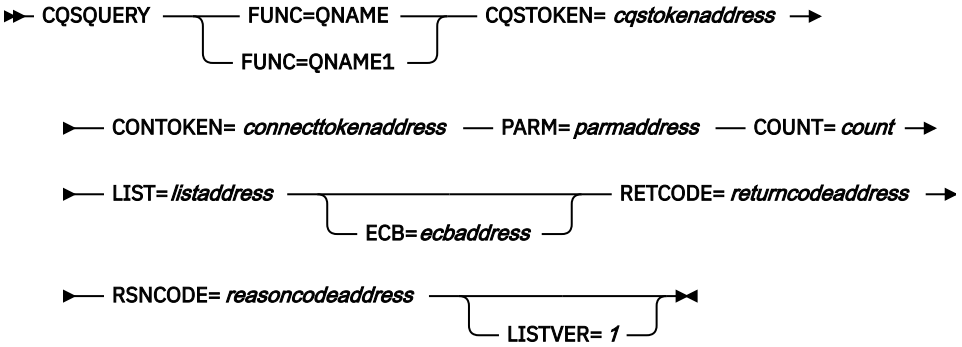
DSECT function of CQSQUERY

You 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.



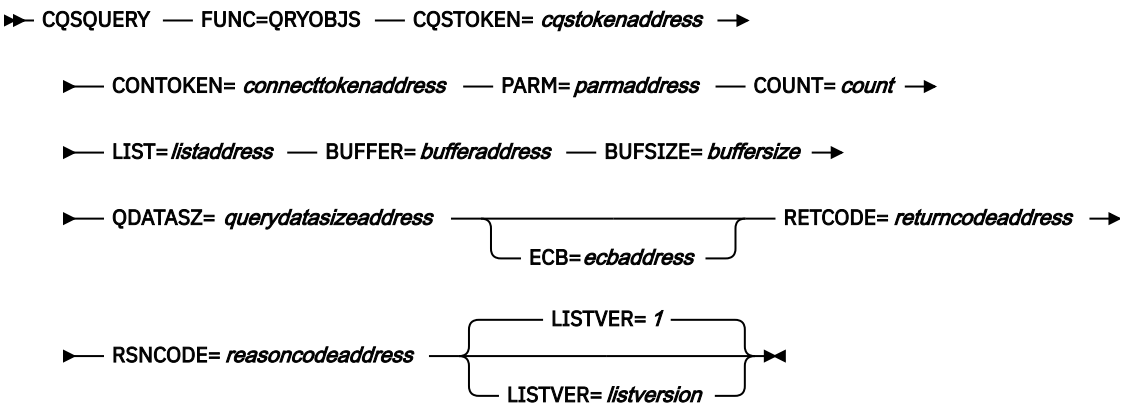
QNAME and QNAME1 function of CQSQUERY

Use the QNAME or QNAME1 function of a CQSQUERY request to retrieve information about a specific queue managed by CQS.



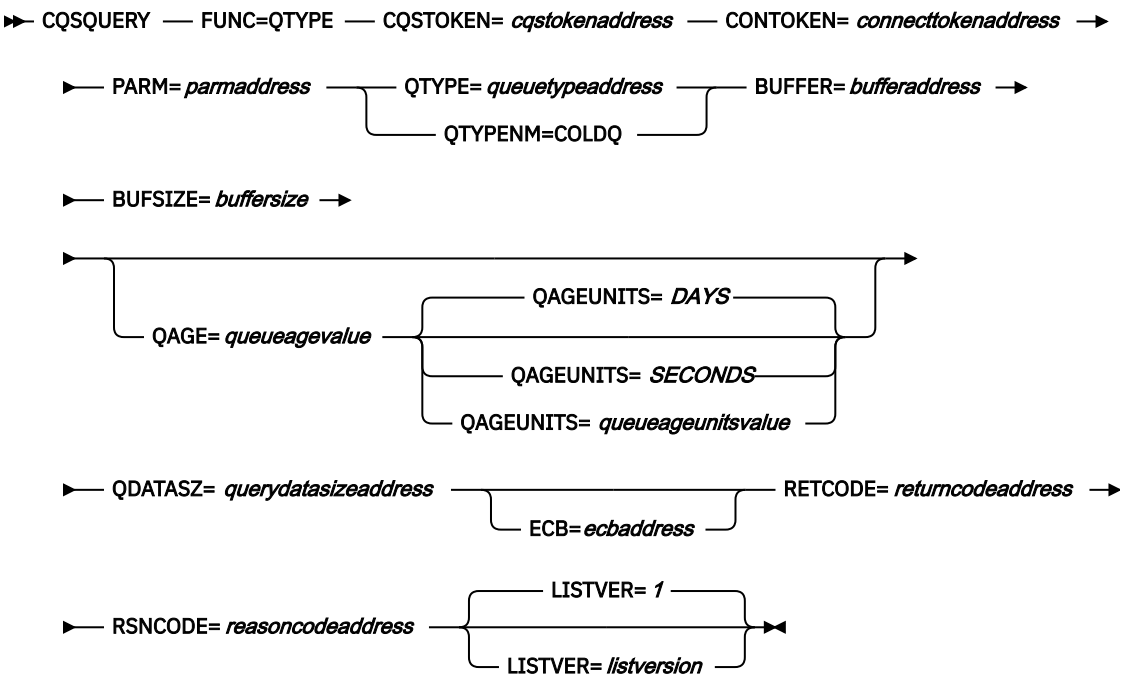
QRYOBJS function of CQSQUERY

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



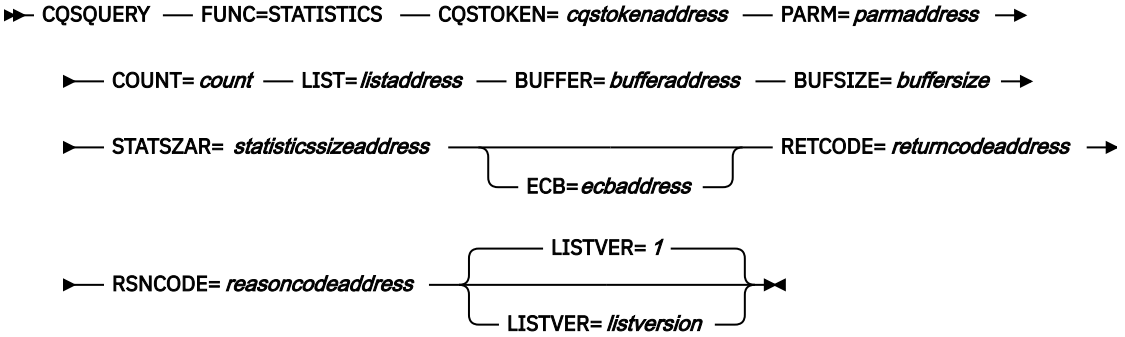
QTYPE function of CQSQUERY

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

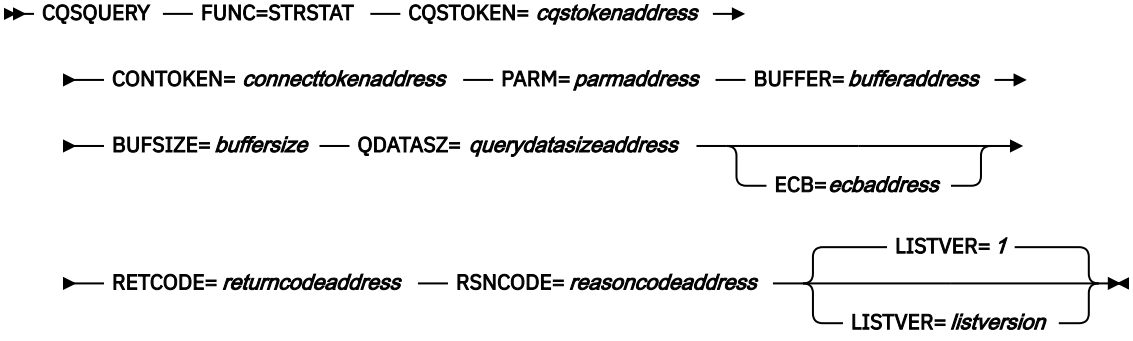
Use the STATISTICS function of a CQSQUERY request to retrieve status information on all the queues managed by CQS.



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 the number of objects on one or more specific queues managed by CQS.
- A `CQSQUERY FUNC=QNAME1` retrieves information about whether a queue is empty or not for one or more specific queues managed by CQS.
- A `CQSQUERY FUNC=QRYOBJS` 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=QNAME1, CQSQUERY FUNC=QRYOBJS, and CQSQUERY FUNC=QTYPE requests are not supported for resource structures.

Below is a list of CQSQUERY requests:

CQSQUERY FUNC=*QNAME*
For CQSQUERY FUNC=QNAME, the number of data objects for the *queuename* specified in LIST= is returned.

CQSQUERY FUNC=*QNAME1*
For CQSQUERY FUNC=QNAME1, the status of each queue is returned as either being empty or containing at least one object for the *queuenames* specified in LIST=.

FUNC=QNAME1 is identical to FUNC=QNAME except for the outputs. FUNC=QNAME1 requires fewer coupling facility (CF) accesses than FUNC=QNAME when there are many objects on a queue. It can be used for situations where users only need to know whether the queue is empty or not.

Important: CQSQUERY FUNC=QNAME1 is available in IMS V15 and higher versions and requires APAR PH40806.

If a CQSQUERY FUNC=QUNAME1 request is issued to a CQS that is either lower than IMS V15, or is at IMS V15 but without APAR PH40806 installed, the request fails with either:

- For authorized clients, return code `RQRCPARM (X'08')`, reason code `RQYFUNC (X'0218')`
- For non-authorized clients, return `code RQRCSYS (X'14')`, reason code `RSCBDRQF (X'0628')`

Code using CQSQUERY FUNC=QNAME1 can tolerate a back-level CQS by checking for the above return/reason code combinations, and issuing CQSQUERY FUNC=QNAME if either is received.

CQSQUERY FUNC=*QRYOBJS*
For CQSQUERY FUNC=QRYOBJS, 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=QRYOBJS 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=QRYOBJS 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.

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=*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: The client program can then make another request with a larger buffer.

- 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 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 CQSQRYQT DSECT. For CQSQUERY FUNC=STATISTICS, the buffer is mapped by the CQSQRYST DSECT. For CQSQUERY FUNC=STRSTAT, the buffer is mapped by the CQSQSTAT DSECT. For CQSQUERY FUNC=QRYOBJS, the buffer is mapped by the CQSQRYQO DSECT.

BUFSIZE=*buffersize*
Four-byte input parameter that specifies the size of the buffer passed by the client.

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

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

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

ECB=*ecbaddress*
Four-byte input parameter that specifies the address of the 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, CQSQUERY FUNC=QNAME1 and CQSQUERY FUNC=QRYOBJS requests, this list contains queue names for which to retrieve information. The list consists of input and output parameters. At least one list item is required.

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

- For a CQSQUERY FUNC=QNAME or CQSQUERY FUNC=QNAME1 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 field to receive one of the following output information about the queue when the request is successfully completed (the compcode field is X'00000000'):

- For FUNC=QNAME, qcnt contains the count of objects on the queue.
- For FUNC=QNAME1, qcnt contains zero if there are no objects on the queue. It contains a non-zero value if there is at least one object on the queue.

- For a CQSQUERY FUNC=STATISTICS request, each list entry contains the following parameters:

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

X'00000000'
Request completed successfully.

X'00000008'
connecttoken is invalid.

X'0000000C'
A CQSRSYNC is required for this structure.

X'00000020'
Structure is inaccessible. Retry request.

X'00000024'
CQS internal error.

clientdata
Eight-byte input parameter that specifies the client data field. This parameter is optional. CQS does not use data stored in this entry.

connecttoken
Sixteen-byte input parameter that specifies the connect token that uniquely identifies the client's connection to a particular coupling facility structure managed by this CQS. The connect token is returned by the CQSCONN request. This parameter is required.

outputoffset
Four-byte output parameter that specifies the offset of the output data area for this entry in the output buffer.

- For a CQSQUERY FUNC=QRYOBJS request, each list entry contains the following parameters:

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=QRYOBJS to obtain the queue counts. The output returned in the output buffer might be invalid.

X'0024'
CQS internal error. Retry the CQSQUERY FUNC=QRYOBJS to obtain the queue counts. The output returned in the output buffer might be invalid.

clientdata
Eight-byte input parameter that specifies the client data field. This parameter is optional. CQS does not use data stored in this entry.

queuename
Sixteen-byte input parameter that specifies the queue name for which data object count information is to be retrieved. This parameter is required. The queuename can be a wildcard parameter.

qcnt
Four-byte output parameter that specifies a field to contain the data object count for the queue name specified. If the queuename is a wildcard parameter, this parameter specifies a field to contain the total queue counts of all qnames that match the wildcard parameter.

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

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

QAGE=*queueagevalue*

Input parameter that specifies the address of a 4-byte field to contain the queue age in the unit that is specified by the **QAGEUNITS** parameter.

The queue age is determined by the age of its oldest message. Valid queue age values are:

- From X'0' to X'16D' (0 to 365 in decimal) if the unit is days
- From X’0’ to X’ 1E13380’(0 to 31536000 in decimal) if the unit is seconds

This parameter is used as a filter for determining which queues the CQSQUERY FUNC=QTYPE request processes. The CQSQUERY request returns information for queues that contain data objects which are older than the specified *queueagevalue*. If you specify 0 for *queueagevalue*, 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.

QAGEUNITS=DAYS|SECONDS|queueageunitsvalue
Input parameter that specifies the unit of the queue age value that is passed onto QAGE. Available units are DAYS, SECONDS, or a value that can be set at run time in a register or in a 4-byte field in storage. The default unit of the queue age is DAYS.

If the **QAGEUNITS** parameter is specified as SECONDS or the value is 1 (EQU value QRY_QAGEUNITS_SECONDS, defined in macro CQSQRYL), the unit of the queue age is seconds.

If **QAGEUNITS** is specified as DAYS or the value is 0 (EQU value QRY_QAGEUNITS_DAYS, defined in macro CQSQRYL), the unit of the queue age is days.

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

The following table 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 17. CQSQUERY return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000004'	X'00000120'	The buffer size (*buffersize*) is less than the query-data size (*querydatasize*). Partial data is returned. *querydatasize* points to the actual buffer size needed to contain all the data.
X'00000004'	X'00000124'	*buffersize* is too small to contain data for number of entries specified in *list*.
X'00000004'	X'00000128'	No data objects on queue type.
X'00000008'	X'00000210'	*cqstoken* is invalid.
X'00000008'	X'00000214'	*connecttoken* is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000224'	*bufferaddress* is invalid.
X'00000008'	X'00000228'	*buffersize* is invalid.
X'00000008'	X'0000022C'	*statisticssize* or *querydatasize* is invalid.
X'00000008'	X'0000023C'	*queueage* is invalid.
X'00000008'	X'00000240'	*queuetype* is invalid.
X'00000008'	X'00000250'	*count* is invalid.
X'00000008'	X'00000254'	*listaddress* is invalid.
X'00000008'	X'0000027C'	CQSQUERY FUNC=QNAME, CQSQUERY FUNC=QNAME1, CQSQUERY FUNC=QTYPE, or CQSQUERY FUNC=QOBJS 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 *compcode* for individual errors.

Table 17. CQSQUERY return and reason codes (continued)

Return code	Reason code	Meaning
X'0000000C'	X'00000304'	Request failed for all list entries. See *compcode* 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.

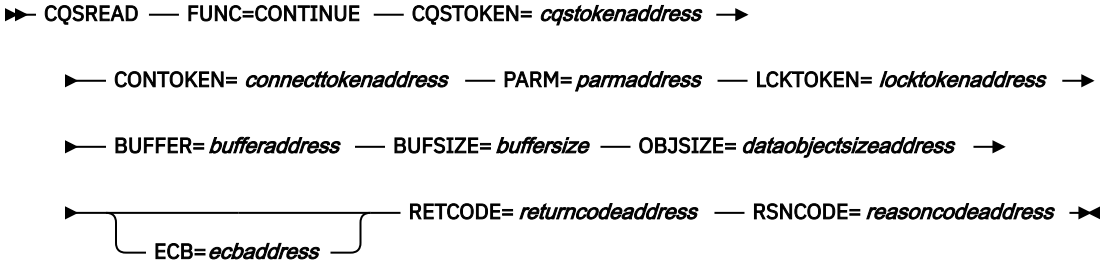
CQSREAD request

The CQSREAD request retrieves a copy of the client data object from a specific queue.

Format for CQSREAD

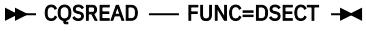
CONTINUE function of CQSREAD

You 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.



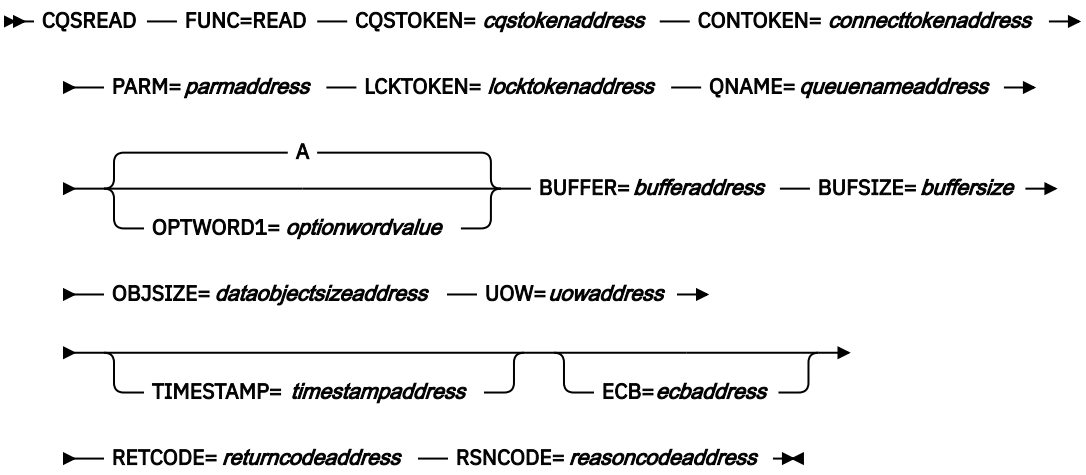
DSECT function of CQSREAD

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.

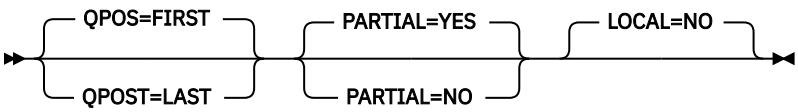


READ function of CQSREAD with LOCAL=NO

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.

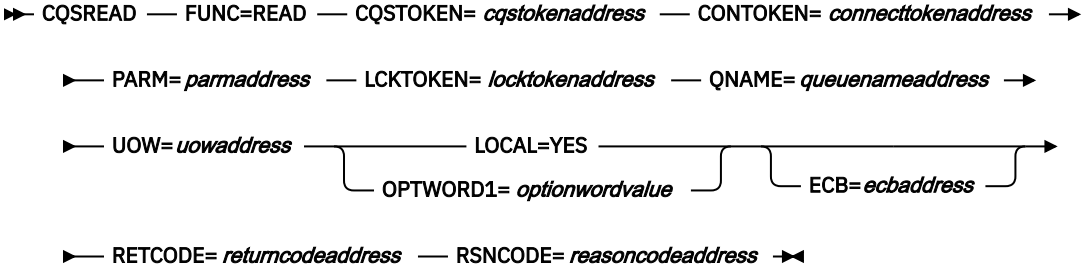


A



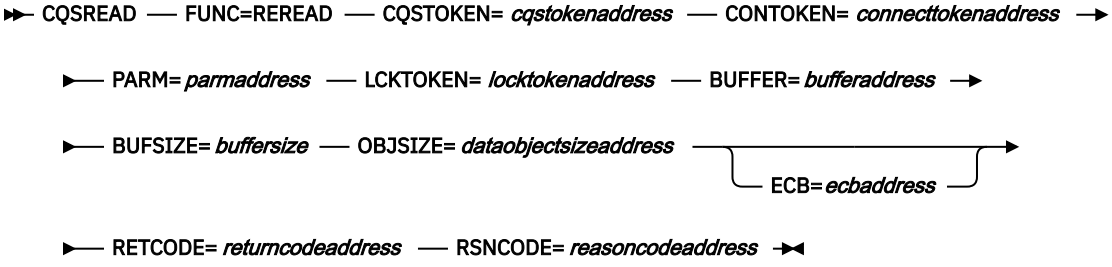
READ function of CQSREAD with LOCAL=YES

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.



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.



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, preventing 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 the following 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_PRTLYEQUX   PARTIAL=YES
```

QNAME=*queuenameaddress*
Input parameter that specifies the address of the 16-byte queue name from which the data object is to be retrieved. The first byte of the queue name identifies the queue type.

QPOS=FIRST | LAST
Input parameter that specifies the position on the queue from which the data object is to be retrieved.

FIRST
The data object is retrieved from the beginning of the queue.

LAST
The data object is retrieved from the end of the queue.

The QPOS parameter cannot be used when the OPTWORD1 parameter is specified. If the OPTWORD1 parameter is specified instead of QPOS, you can use the following equate (EQU) statements to generate the value for the OPTWORD1 parameter:

```
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 time stamp 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 time stamp 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

The following table 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 18. CQSREAD return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000004'	X'00000120'	The buffer size (*buffersize*) is less than the data object size (*dataobjectsize*). Partial data is returned. *dataobjectsize* contains the address of the actual data object size.

Table 18. CQSREAD return and reason codes (continued)

Return code	Reason code	Meaning
X'00000004'	X'00000124'	The buffer size (*buffersize*) is less than the data object size (*dataobjectsize*). No data is returned because PARTIAL=NO was specified. *dataobjectsize* contains the address of the actual data object size.
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'	*cqstoken* is invalid.
X'00000008'	X'00000214'	*connecttoken* is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'0000021C'	*locktoken* is invalid.
X'00000008'	X'00000220'	*queuename* is invalid.
X'00000008'	X'00000224'	*bufferaddress* is invalid.
X'00000008'	X'00000228'	*buffersize* is invalid.
X'00000008'	X'0000022C'	*dataobjectsize* is invalid.
X'00000008'	X'00000230'	*uow* 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.
X'00000014'	X'00000504'	z/OS logger write error, data objects were not retrieved from queues.

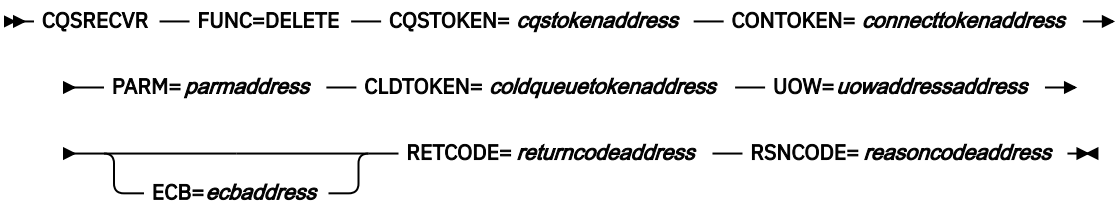
CQSRECVR request

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.

Format for CQSRECVR

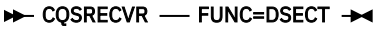
DELETE function of CQSRECVR

Use the DELETE function of a CQSRECVR request to delete one data object associated with a UOW from the cold queue.



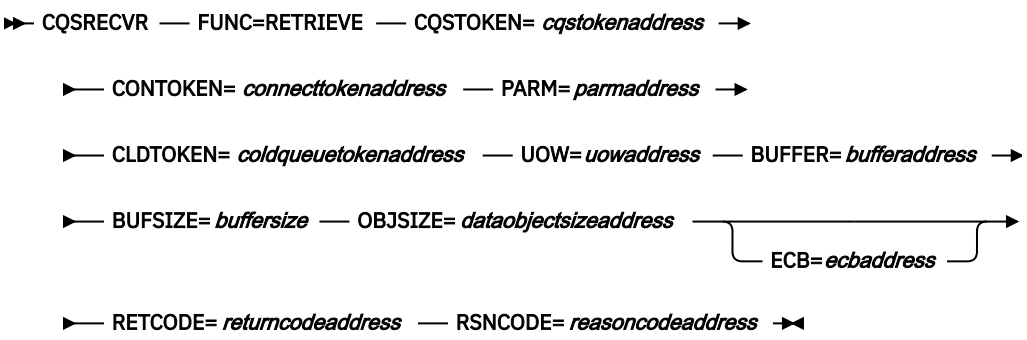
DSECT function of CQSRECVR

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



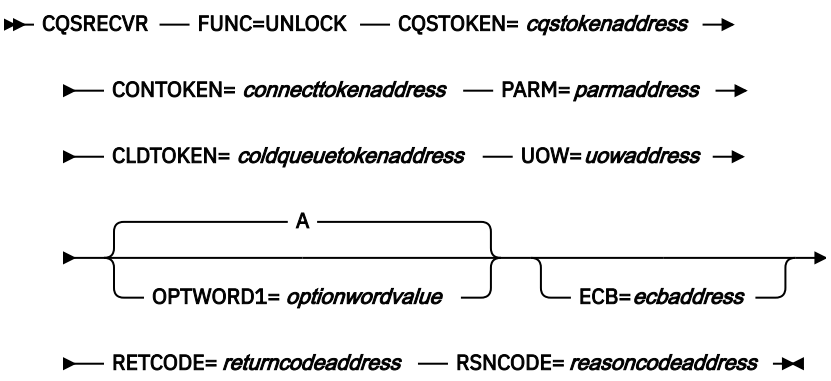
RETRIEVE function of CQSRECVR

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

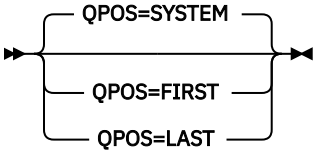


UNLOCK function of CQSRECVR

Use the UNLOCK function of a CQSRECVR request to unlock a data object associated with a UOW on the cold queue.



A



Usage of CQSRECVR

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=*dataobjectsizeaddress*
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

The following table shows 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 19. CQSRECVR return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000004'	X'00000124'	*buffersize* is too small.
X'00000004'	X'00000128'	Data object for UOW not found on cold queue.
X'00000008'	X'00000210'	*cqstoken* is invalid.

Table 19. CQSRECVR return and reason codes (continued)

Return code	Reason code	Meaning
X'00000008'	X'00000214'	*connecttoken* is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000224'	*bufferaddress* is invalid.
X'00000008'	X'00000228'	*buffersize* is invalid.
X'00000008'	X'0000022C'	*dataobjectsize* is invalid.
X'00000008'	X'00000230'	*uow* is invalid.
X'00000008'	X'00000234'	*coldqueuetoken* 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

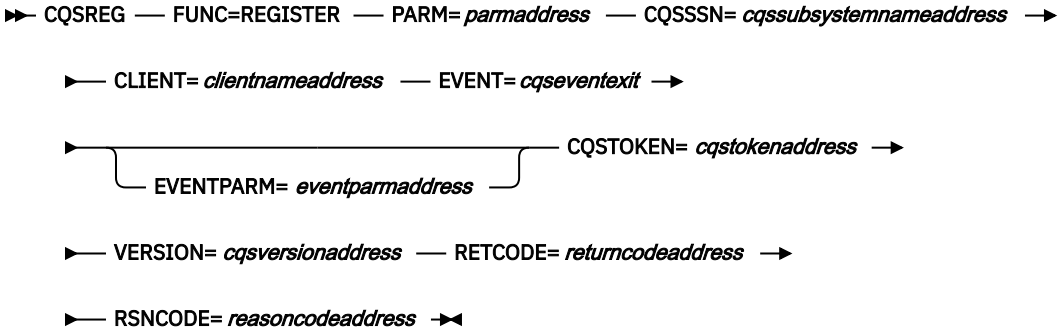
The CQSREG request registers a client to CQS.

Format for CQSREG

DSECT function of CQSREG

You 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.

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



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.

CLIENT=*clientnameaddress*
Input parameter that specifies the address of the 8-byte name of the client registering to CQS. The client name must be unique among all clients that are registered to the same CQS and to all the CQSs that are sharing the same queues.

CQSTOKEN=*cqstokenaddress*
Output parameter that specifies the address of a 16-byte area to receive the CQS registration token that uniquely identifies the client's connection to CQS. The registration token is returned by a successful CQSREG request.

CQSSSN=*cqssubsystemnameaddress*
Input parameter that specifies the address of the 4-byte subsystem name of the CQS to which the client would like to connect. This parameter should match the SSN= parameter of the CQSIPxxx PROCLIB member for the CQS to which the client would like to connect.

EVENT=*cqseventexit*
Four-byte input parameter that specifies the CQS Event exit routine address.

EVENTPARM=*eventparmaddress*
Input parameter that specifies the address of a 4-byte area that contains client data that CQS passes to the CQS Event exit routine every time the exit is called.

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

RETCODE=*returncodeaddress*
Output parameter that specifies the address of a 4-byte field to contain the CQSREG return code. The CQSREG return code is returned both in this field and in register 15.

RSNCODE=*reasoncodeaddress*
Output parameter that specifies the address of a 4-byte field to contain the CQSREG reason code. The CQSREG reason code is returned both in this field and in register 0.

VERSION=*cqsversionaddress*
Output parameter that specifies the address of a 4-byte area to receive the CQS version number. The version number has the following format: `00vvrrmm`.

00
This byte is reserved for future use. Currently, it is always 00.

vv
Version number.

rr
Release number.

mm
Modification level or sub-release number.

For example, CQS version 1.1.0 is shown as X'00010100'.

Return and reason codes for CQSREG

The following table lists the return and reason code combinations that can be returned for CQSREG requests.

Table 20. 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'	*clientname* is invalid.
X'00000008'	X'00000284'	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'00000010'	X'00000448'	A registered client address space attempted to register with CQS a second time.
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.
X'00000014'	X'0000053C'	Unable to load CQS registration module CQSREG00.

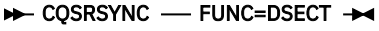
CQSRSYNC request

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.

Format for CQSRSYNC

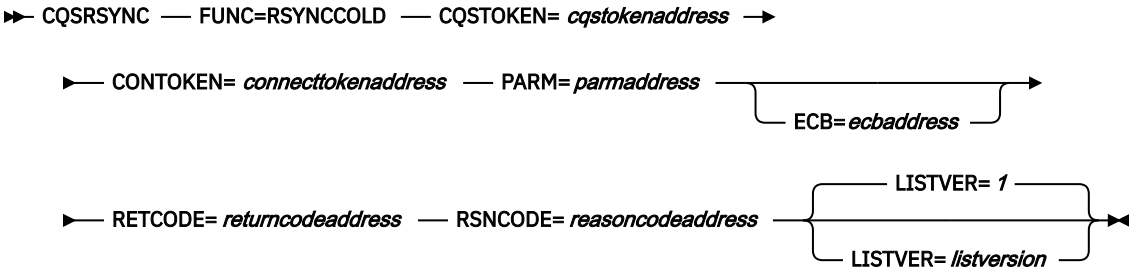
DSECT function of CQSRSYNC

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



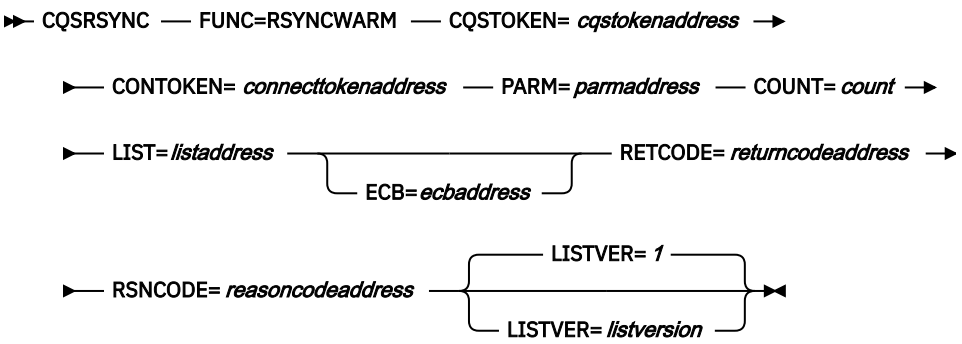
RSYNCCOLD function of CQSRSYNC

Use the RSYNCCOLD function of a CQSRSYNC request when the client is performing a cold start and does not have information on unresolved UOWs.



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 the following table.

Table 21. 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 the following table.

Table 22. 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.
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.

Table 22. UOW status from CQS (continued)

Status	Meaning
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: • 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

The following table 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 23. 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.
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.

Table 23. CQSRSYNC return and reason codes (continued)

Return code	Reason code	Meaning
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'	*cqstoken* 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'	*connecttoken* 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'	*listaddress* 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 *compcode* in each list entry for individual errors. The Structure Event exit routine is called for CQS resync. The client can now issue CQS requests to write or retrieve data for this structure.
X'0000000C'	X'00000304'	Request failed for all list entries. None of the list entries are in sync. See *compcode* in each list entry for individual errors. The Structure Event exit routine is called for CQS resync. The client can now issue CQS requests to write or retrieve data for this structure.
X'00000010'	X'00000430'	No CQS address space.
X'00000014'	X'00000500'	CQS internal error.

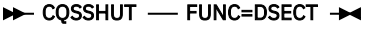
CQSSHUT request

A CQSSHUT request notifies CQS to terminate after all clients have disconnected.

Format for CQSSHUT

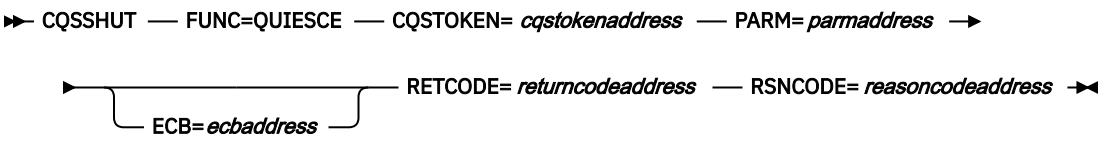
DSECT function of CQSSHUT

You 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.



QUIESCE function of CQSSHUT

Use the QUIESCE function of a CQSSHUT request to terminate CQS.



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.

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

RETCODE=*returncodeaddress*
Output parameter that specifies the address of a 4-byte field to contain the CQSSHUT return code.

If the return code in register 15 is nonzero, the values in the return and reason code fields are invalid, because the CQS interface detected an error and was unable to send the request to CQS.

RSNCODE=*reasoncodeaddress*
Four-byte output parameter that specifies the address of a field to contain the CQSSHUT reason code.

Return and reason codes for CQSSHUT

The following table 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 24. CQSSHUT return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000008'	X'00000210'	*cqstoken* 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.

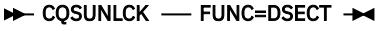
CQSUNLCK request

A CQSUNLCK request unlocks one or more data objects and moves them into the first or last position on the queue. You can also force an unlock by specifying FUNC=FORCE.

Format for CQSUNLCK

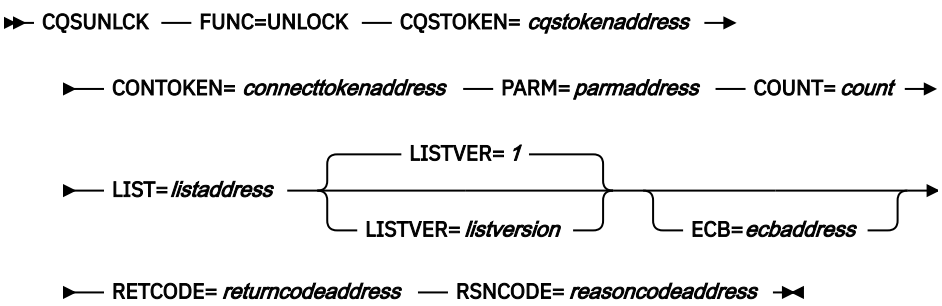
DSECT function of CQSUNLCK

You 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.



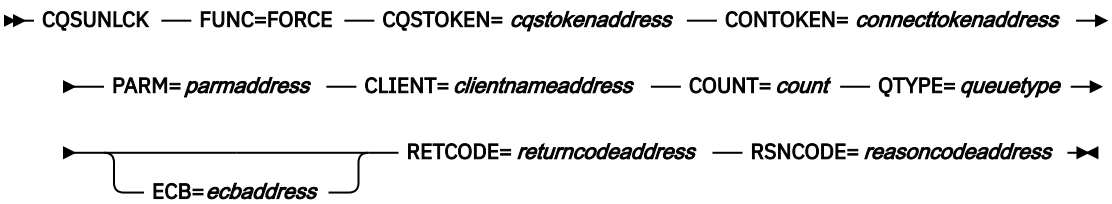
UNLOCK function of CQSUNLCK

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.



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 do not 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 do not 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 ensure that the target CQS client is not active.

  It is up to the CQS clients in the IMSplex to ensure that only one CQS client issues the CQSUNLCK FUNC=FORCE request. All members in an IMSplex defined with a CSL are notified when a member fails. Multiple CQSUNLCK FUNC=FORCE requests may have the following undesirable results:

  – Unnecessary CF accesses.

    The CQSUNLCK FUNC=FORCE request incurs multiple CF accesses to look at data objects on the candidate queue type. If multiple CQSUNLCK FUNC=FORCE requests are issued, each request makes the same numerous CF accesses. These extra CF accesses are unnecessary and incur additional performance overhead. If the performance overhead of unnecessary CF accesses is unacceptable, it is up to the CQS clients in the IMSplex to ensure that only one CQS client issues the CQSUNLCK FUNC=FORCE.

    It is up to the CQS clients in the IMSplex to ensure 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 ensure 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 ensure that exactly one CQS client forcibly unlocks data objects for the specified client.

- Forcibly unlock CQS client's own data objects when CQS client initializes.

  When a CQS client initializes, it should forcibly unlock its own data objects before issuing CQSRSYNC. This ensures that the CQS client's data objects are unlocked before resync, in case no other CQS client was available at failure time to do the force unlock. Force unlock also cleans up CQS's knowledge of the IMS client's locked data objects, since this CQS processed the CQSREAD request that locked the data objects.

- Resync with CQS, handling UOWs 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. Retry request later.

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.

X'00000018'
z/OS logger write error, data objects were not unlocked.

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

The following table 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 25. CQSUNLCK return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000008'	X'00000210'	*cqstoken* is invalid.
X'00000008'	X'00000214'	*connecttoken* is invalid.

Table 25. CQSUNLCK return and reason codes (continued)

Return code	Reason code	Meaning
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000240'	*queuetype* is invalid.
X'00000008'	X'00000244'	*clientname* is invalid.
X'00000008'	X'00000250'	*count* is invalid.
X'00000008'	X'00000254'	*listaddress* is invalid.
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 *compcode* for individual errors.
X'0000000C'	X'00000304'	Request failed for all list entries. See *compcode* for individual errors.
X'00000010'	X'00000400'	A CQSRSYNC is required for this structure.
X'00000010'	X'00000430'	No CQS address space.

CQSUPD request

The CQSUPD request creates or updates one or more uniquely named resources on a resource structure. The CQSUPD request creates a resource if it does not exist, or updates a resource if it does exist.

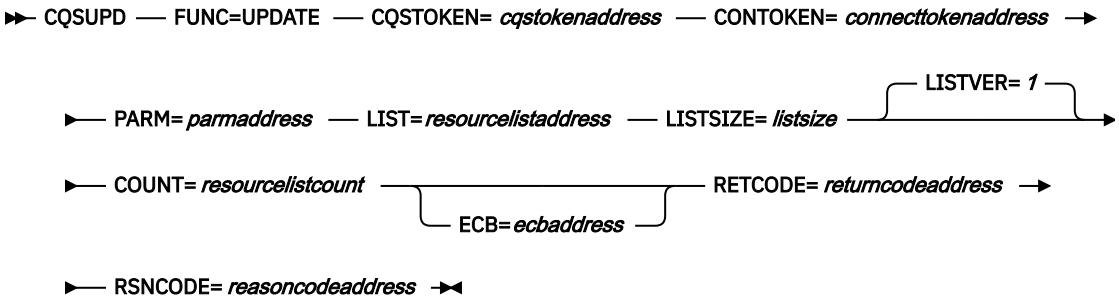
Format for CQSUPD

DSECT function of CQSUPD

You 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.

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).



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

The following 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 26. CQSUPD return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000008'	X'00000210'	*cqstoken* is invalid.
X'00000008'	X'00000214'	*contoken* is invalid.
X'00000008'	X'00000218'	FUNC is invalid.
X'00000008'	X'00000250'	*resourcelistcount* is invalid.
X'00000008'	X'00000254'	*listaddress* 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.

Table 26. CQSUPD return and reason codes (continued)

Return code	Reason code	Meaning
X'0000000C'	X'00000300'	Request succeeded for at least one but not all list entries. See *compcode* for individual errors.
X'0000000C'	X'00000304'	Request failed for all entries. See *compcode* for individual errors.
X'0000000C'	X'00000308'	Request failed for one or more list entries because of version mismatch. Those resources already exist as the *resourcetype* specified. All other entries were successful.
X'00000010'	X'00000430'	No CQS address space.
X'00000014'	X'00000500'	Internal error.

Part 2. Common Service Layer (CSL)

The topics included in this information provide information about the CSL.

Chapter 3. Writing a CSL client

These topics describe the considerations for writing a CSL client. This information is for the programmer who writes the client, but also for the CSL administrator or IMS system programmer who must be aware of the issues involved in designing and writing a CSL client.

Event Control Blocks with CSL requests

The event control block (ECB) is an optional parameter within a CSL request that allows you to specify the address of the z/OS ECB.

Most CSL requests allow an ECB to be specified. The ECB parameter is optional and specifies the address of the z/OS ECB. When a CSL request completes, the ECB specified by the ECB parameter is posted. If the parameter is not included, the requesting module does not receive control until the request completes.

If an ECB is specified, the invoker of the request must issue a WAIT (or equivalent) after receiving control from the request before the invoker uses or examines any data that is returned by this request (including the RETCODE and RSNCODE fields). If the WAIT is not issued, the data might be invalid.

Environmental requirements for SCI requests

The environmental requirements for SCI requests depend on the SCI interface that is assigned to the client.

The following table describes the environment for authorized SCI requests.

Table 27. Environment for SCI requests that use the authorized SCI interface

Environmental characteristic	**Requirement**
Authorization	Supervisor state (PSW key must match the PSW key when the CSLSCREG request was issued)
Dispatchable unit mode	Task
Cross memory mode	Any, however, PASN must equal the primary address space where the CSLSCREG 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

The following table describes the environment for non-authorized SCI requests.

Table 28. Environment for SCI requests using the non-authorized interface

Environmental characteristic	**Requirement**
Authorization	Problem state (PSW key must match the PSW key when the CSLSCREG request was issued)
Dispatchable unit mode	Task

Table 28. Environment for SCI requests using the non-authorized interface (continued)

Environmental characteristic	Requirement
Cross memory mode	None (PASN=SASN=HASN)
AMODE	31
ASC Mode	Primary
Home address space	Address space where CSLSCREG was issued
Locks	No locks held
Interrupt status	Enabled for interrupts
Control parameters	In primary address space

The environmental requirements for the SCI register and deregister requests (CSLSCREG and CSLSCDRG) are different from all of the other SCI requests. Authorized clients must issue CSLSCREG and CSLSCDRG requests in the environment shown in the following table:

Table 29. Environment for CSLSCREG and CSLSCDRG requests using the authorized interface

Environmental Characteristic	Requirement
Authorization	Supervisor state
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 CSLSCREG and CSLSCDRG requests in the environment described in the following table:

Table 30. Environment for CSLSCREG and CSLSCDRG requests using the non-authorized interface

Environmental Characteristic	Requirement
Authorization	Problem state
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

How to interpret CSL request return and reason codes

Common Service Layer (CSL) return and reason codes indicate the success or failure of sending the request to the CSL address space and reflect the success or failure of the particular CSL request that is being made.

Each of the CSL requests receives a return and reason code that indicates the result of the specified request. Because most of the requests involve more than one component, return and reason codes can originate from any of the components involved. For example, a Structured Call Interface (SCI) return and reason code can be received because SCI is the communications mechanism for these requests. The high order byte is used to help identify the component that set the return and reason codes.

The possible values of the high-order byte and the meanings of each value are:

X'00'
IMS set the return and reason code

X'01'
SCI set the return and reason code

X'02'
Operations Manager (OM) set the return and reason code

X'03'
Resource Manager (RM) set the return and reason code

X'04'
ODBM set the return and reason code

Each of the CSL requests have a table of return and reason codes. If you are unable to find the return and reason code for the request that you issued, use the high order byte in the return code to help identify the component that set the return and reason code. For example, if the reason code is X'01' (SCI), you should start by looking at the return and reason codes for the CSLSCMSG and CSLSCRQS macros.

ODBM reason codes are defined in the CSLDRR macro, OM reason codes are defined in the CSLORR macro, RM reason codes are defined in the CSLRRR macro, and SCI reason codes are defined in the CSLSRR macro. These macros can be found in the IMS.SDFSMAC data set.

Related reference

“CSLZQRY: query request” on page 102
In an IMSplex, you might want to query statistics about one or more components in the CSL. You can write an IMSplex member program, for example, an automated operator program (AOP), that uses the CSLZQRY request to obtain statistics. Any member of an IMSplex can issue the CSLZQRY request.

“CSLZSHUT: shutdown request” on page 104
CSLZSHUT is a programming interface that enables you to shut down one or more CSL components from an authorized IMSplex member. Because CSLZSHUT is sent as a message, control is returned to the program that issued the request after the message is sent.

Planning considerations for writing clients for the CSL

Planning tasks are decisions that you must make to determine how you use the CSL managers and CSL requests.

Planning tasks are decisions that you must make to determine how you use the CSL managers. These decisions include:

- **What authorization level to use**

  You must decide whether your program needs to run authorized (supervisor state) or non-authorized (problem state). SCI initializes the appropriate environment based on your program's state and PSW key when it registers with SCI.

  Note: A non-authorized client cannot register with RM, issue RM requests, register commands with OM, or process requests issued using CSLSCRQS.

- **Whether to use SCI exit routines**

  You must decide whether to use the SCI Input and Notify exit routines. An OM command processing client, for example, must have the SCI Input exit to process OM directives; it must have the SCI Notify exit to be notified when new OMs join the IMSplex, so the OM command processing client can register to those OMs.

- **TCB association**

  SCI registration (with the CSLSCREG request) enables an IMSplex member to be associated with a specific, different TCB. The authorization level you use must also be considered regarding TCB association. SCI internally associates the registration with the specified TCB. If no TCB is specified, SCI associates the registration with the TCB from which the registration is issued. If the associated TCB terminates without a deregistration being issued, SCI abnormally terminates the registration and releases the associated storage that SCI allocated in the member address space. If a subsequent SCI request is issued, an abend may occur.

- **Whether to use RM services, OM services, or ODBM services**

  You can choose to manage your own global resources. However, if you want to access IMS global resources, you must code an RM client.

  If you plan to develop your own command set and your own command processing client (which would coordinate its own command registration and security), you can write an OM command processing client. If you plan to develop your own SPOC or AOP to enter your own commands, you can write an OM AOP client. OM's role is to transport commands throughout an IMSplex and to consolidate those command responses, in XML tags, for a SPOC or AOP.

  For access to IMS databases managed by IMS DB in either the DBCTL or DB/DC environment, you can write an ODBM client application program that does not use IMS transactions. ODBM manages database connections in an IMSplex, permitting application programs that use either the IMS Universal drivers or the ODBA interface to access databases in an IMSplex. ODBM also protects the IMS control region from the unexpected termination of application programs that use the ODBA interface.

- **Whether to use message or request protocol when issuing requests**

  Use message protocol either when you do not need a synchronous response, or when you want an asynchronous response. IMSplex command responses that are sent with message protocol are sent asynchronously.

- **Whether to use the CSL OM audit trail**

  The CSL OM audit trail contains normal operating messages generated by active CSL address spaces including RM, OM, and SCI, as well as operational messages created by IMS components routing activity through a CSL component. CSL client activity is also captured. The audit trail can be used for audit compliance as well as for diagnostic tasks.

- **Whether to use the CSL ODBM accounting feature**

  ODBM leverages the z/OS System Management Facility (SMF) to perform logging of ODBM accounting information, such as CPU usage, and its retrieval. The logging of the ODBM address space is activated when the optional parameter LOGOPT=ACCOUNTING is specified in the ODBM initialization member, CSLDIxxx.

Related concepts

CSL OM audit trail (System Administration)

Related reference

BPE-based CSL SCI user exit routines (Exit Routines)

Registration of CSL managers with SCI

ODBM, OM, and RM clients must register with the Structured Call Interface (SCI). You must complete several registration steps in order to use any of the CSL managers.

To use any of the CSL managers, you must first complete registration steps. ODBM, OM, and RM clients must register with the SCI. This topic describes SCI registration and how ODBM, OM, and RM clients register with the SCI, how to set SCI to a ready state, and the sequence in which CSL requests must be issued.

SCI registration

You must register to SCI in order to uniquely identify an IMSplex member's connection to SCI that is used on all subsequent requests.

When you register to SCI, you identify:

- The name of the IMSplex.
- Your client name, which must be unique if it is an authorized client.
- Exit routines, if you elect to use them.
- Your type of address space.

  Use a type of AOP or OTHER for the address space. Defining your address space by a type that is not AOP or OTHER could interfere with IMS address spaces. You can further identify your client by using the SUBTYPE parameter.

After you register to SCI, an SCI token is returned. The token uniquely identifies an IMSplex member's connection to SCI and is used on all subsequent requests. Save the token for future ODBM, OM, RM, and SCI requests.

Registering an ODBM client

To register with ODBM, a client must first register with the CSL SCI and then with all active ODBMs in the IMSplex.

Application servers for application programs that use the IMS ODBA interface to access IMS databases can register with ODBM as a client. ODBM manages connections to databases owned by IMS DB systems in an IMSplex and protects IMS control regions from the unexpected termination of the application program.

To register an ODBM client:

1. Identify the exit routines that are needed to drive interactions between the CSL SCI and the client.

   The SCI handles communications between all CSL managers in the IMSplex and their client applications.

   a) Identify the SCI Notify exit for the client.

      A database connection can be routed through any active ODBM in the IMSplex, so an ODBM client must registered to all ODBMs. A SCI Notify exit is driven when a new ODBM becomes active in the IMSplex and notifies the client with information about the new ODBM. The client can then register to the new ODBM. The SCI Notify exit is identified with the NOTIFYEXIT parameter of the CSLSCREG request.

   b) Identify the SCI Input exit for the client with the INPUTEXIT parameter of the CSLSCREG request.

      An ODBM client must be able to receive ODBM directives sent from any ODBM in the IMSplex. The SCI Input exit is driven when there is a message or request for the client, so that the client can receive and process it.

2. Issue the CSLSCREG request to SCI with the information gathered in step 1.

   A client must register to the CSL SCI in the IMSplex before it can register with a CSL manager such as ODBM.

3. Issue the CSLSCQRY request to SCI to determine which instances of ODBM are active in the IMSplex.

Before registering with the ODBMs for the first time, the client must manually identify all active ODBMs with this request. Once registered, the SCI Notify exit automatically informs the client when new ODBMs become active.

4. Issue the CSLDMREG request to all ODBMs in the IMSplex that are reachable and ready to accept registration requests.

Consult the sequence of ODBM client requests for more information about managing the CSL registrations for an ODBM client.

Related concepts

"Sequence of ODBM client requests" on page 129
Some requests to Open Database Manager (ODBM) from an ODBM client must be issued in a particular sequence, such as when enabling or disabling communication with ODBM.

Related reference

"CSLSCREG: registration request" on page 215
The Structured Call Interface (SCI) registration request is used to create a connection between an IMSplex member and SCI. Before SCI can be used for communication within the IMSplex, an IMSplex member must issue the CSLSCREG request and receive an SCI token when the request completes in order to be recognized by SCI.

"CSLDMREG: ODBM client registration request" on page 141
The CSLDMREG request registers an ODBM client with ODBM.

Registering an OM command processing client

An Operations Manager (OM) command processing client must register its command with OM, whereas automated operator program (AOP) clients do not have to register to OM.

Perform the following steps to register an OM command processing client in an IMSPlex:

1. Identify the exit routines that are needed to drive interactions between the CSL SCI and the client.

   The SCI handles communications between all CSL managers in the IMSplex and their client applications.

   a) Identify the SCI Notify exit for the client.

      A database connection can be routed through any active OM in the IMSplex, so an OM command processing client must registered to all OMs. A SCI Notify exit is driven when a new OM becomes active in the IMSplex and notifies the client with information about the new OM. The client can then register to the new OM. The SCI Notify exit is identified with the NOTIFYEXIT parameter of the CSLSCREG request.

   b) Identify the SCI Input exit for the client with the INPUTEXIT parameter of the CSLSCREG request.

      An OM command processing client must be able to receive OM directives sent from any OM in the IMSplex. The SCI Input exit is driven when there is a message or request for the client, so that the client can receive and process it.

2. Issue the CSLSCREG request to SCI with the information gathered in step 1.

   A client must register to the CSL SCI in the IMSplex before it can register with a CSL manager such as OM.

3. Issue the CSLSCQRY request to SCI to determine which instances of OM are active in the IMSplex.

   Before registering with the OMs for the first time, the client must manually identify all active OMs with this request. Once registered, the SCI Notify exit automatically informs the client when new OMs become active.

4. Issue the CSLOMBLD request to build the command list that will be passed to OM.
5. Issue the CSLOMREG request to all OMs in the IMSplex that are reachable and ready to accept registration requests.
6. Issue the CSLOMRDY request to indicate that the client is ready to begin processing commands from OM.

Consult the sequence of OM command processing client requests for more information about managing the CSL registrations for an OM command processing client.

Related reference

"CSL OM command processing client requests" on page 145

The following topics describe the requests that are made by command processing clients.

Registering an RM client

Register RM clients to manage resources and access IMSplex-wide processes.

The following steps describe how to register an RM client first with the IMSPlex Structured Call Interface (SCI), and then with the RMs active in the IMSPlex.

1. Identify the exit routines that are needed to drive interactions between the CSL SCI and the client.

   The SCI handles communications between all CSL managers in the IMSplex and their client applications.

   a) Identify the SCI Notify exit for the client.

      A database connection can be routed through any active RM in the IMSplex, so an RM client must registered to all RMs. A SCI Notify exit is driven when a new RM becomes active in the IMSplex and notifies the client with information about the new RM. The client can then register to the new RM. The SCI Notify exit is identified with the NOTIFYEXIT parameter of the CSLSCREG request.

   b) Identify the SCI Input exit for the client with the INPUTEXIT parameter of the CSLSCREG request.

      An RM client must be able to receive RM directives sent from any RM in the IMSplex. The SCI Input exit is driven when there is a message or request for the client, so that the client can receive and process it.

2. Issue the CSLSCREG request to SCI with the information gathered in step 1.

   A client must register to the CSL SCI in the IMSplex before it can register with a CSL manager such as RM.

3. Issue the CSLSCQRY request to SCI to determine which instances of RM are active in the IMSplex.

   Before registering with the RMs for the first time, the client must manually identify all active RMs with this request. Once registered, the SCI Notify exit automatically informs the client when new RMs become active.

4. Issue the CSLRMREG request to all RMs in the IMSplex that are reachable and ready to accept registration requests.

   To manage global resources, register the resource type and associated name type.

How to enable SCI ready state

You use the CSLSCRDY request to enable an IMSplex member to receive messages and requests that are routed by type.

With the SCI, there are two states: registered and ready. The CSLSCRDY request enables an IMSplex member to receive messages and requests routed by type. An IMSplex member that is registered but has not issued a CSLSCRDY request can process only messages and requests that are specifically directed to it.

Sequence for coding CSL requests

Most Common Service Layer (CSL) requests must be issued in a certain sequence.

For more information about the sequence of issuing requests from various CSL clients, see the table in each of the following topics:

- "How AOP clients that run on the host communicate with the CSL OM" on page 125
- "How AOP clients that run on a workstation communicate with the CSL OM" on page 126
- "Processing AOP commands with a command processing client" on page 127

- “Sequence of RM client requests” on page 163
- “Sequence of ODBM client requests” on page 129

Related concepts

“CSL OM automated operator program clients” on page 125
OM provides an API interface for application programs that automate operator actions known as automated operator programs (AOP). You can use an AOP to issue commands that are embedded in an OM API request to an OM.

Requests common to all CSL components

Two requests, CSLZSHUT and CSLZQRY, are common requests that can be processed by all CSL components (OM, RM, and SCI).

CSLZQRY: query request

In an IMSplex, you might want to query statistics about one or more components in the CSL. You can write an IMSplex member program, for example, an automated operator program (AOP), that uses the CSLZQRY request to obtain statistics. Any member of an IMSplex can issue the CSLZQRY request.

CSLZQRY syntax

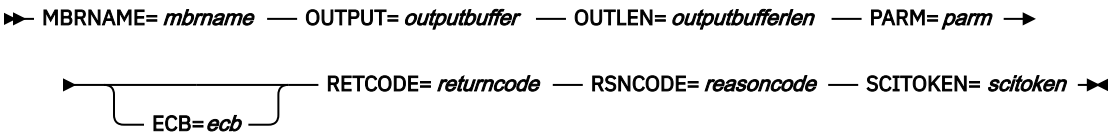
CSLZQRY DSECT syntax

Use FUNC=DSECT to include equate (EQU) statements in your program for the CSLZQRY parameter list length and the CSLZQRY return and reason codes.

CSLZQRY STATS syntax

Use FUNC=STATS to request statistics from ODBM, OM, RM, or SCI. The information that is returned from the CSLZQRY request is the same information that is passed to the STATS exit for that particular ODBM, OM, RM, or SCI.

A



CSLZQRY parameters

ECB=*symbol*
ECB=*(r2-r12)*
(Optional) - Specifies a z/OS event control block (ECB) used for asynchronous requests. When the request is complete, the ECB specified is posted. If an ECB is not specified, the task is suspended until the request is complete. If an ECB is specified, the invoker of the request must issue a WAIT (or equivalent) after receiving control from CSLZQRY and before using or examining any data returned by this request (including the RETCODE and RSNCODE fields).

MBRNAME=*symbol*
MBRNAME=*(r2-r12)*
(Required) - A 4-byte input parameter that specifies the address of the 8-byte CSL member name to which to send the query.

OUTLEN=*symbol*
OUTLEN=*(r2-r12)*
(Required) - A 4-byte output parameter that is used to receive the length of the output buffer. When the request returns, this word contains the length of the buffer pointed to by the OUTPUT= parameter. The output length is zero if no output is built, for example, when an error is detected before any output can be built. When the caller is done with this storage, it is the caller's responsibility to release the storage by issuing a CSLSCBFR request.

OUTPUT=*outputbuffer*
OUTPUT=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the address of the variable length output returned by the CSLZQRY request. The output contains the results of the CSLZQRY. The output length is returned in the OUTLEN= field. The output address is zero if no output was built, for example, if an error was detected before any output could be built. This buffer is not preallocated by the caller. When the caller is done with this storage, it is the caller's responsibility to release the storage by issuing a CSLSCBFR request.

PARM=*symbol*
PARM=*(r2-r12)*
(Required) - Specifies the CSLZQRY parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by ZQRY_PARMLN.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the return code on output. This can be returned by ODBM, OM, RM, or SCI. ODBM return codes are defined in CSLDRR. OM return codes are defined in CSLORR. RM return codes are defined in CSLRRR. SCI return codes are defined in CSLSRR.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. This can be returned by ODBM, OM, RM, or SCI. ODBM reason codes are defined in CSLDRR. OM reason codes are defined in CSLORR. RM reason codes are defined in CSLRRR. SCI reason codes are defined in CSLSRR.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

CSLZQRY return and reason codes

The following table lists the return and reason codes that can be returned on a CSLZQRY macro request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 31. CSLZQRY return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'xx000008'	X'00002050'	The caller of the service attempted to pass an invalid parameter list. The request is rejected. "xx" identifies the component to set the return code.

Related concepts
“How to interpret CSL request return and reason codes” on page 97

Common Service Layer (CSL) return and reason codes indicate the success or failure of sending the request to the CSL address space and reflect the success or failure of the particular CSL request that is being made.

CSLZSHUT: shutdown request

CSLZSHUT is a programming interface that enables you to shut down one or more CSL components from an authorized IMSplex member. Because CSLZSHUT is sent as a message, control is returned to the program that issued the request after the message is sent.

CSLZSHUT allows you to terminate:

- A single CSL manager (ODBM, OM, RM, or SCI)
- A CSL and all of its components on a single z/OS image
- A CSL and all of its components for an IMSplex across multiple z/OS images

The CSLZSHUT request is sent as a message, so control returns to the program that issued the request after the request is sent.

To shut down a single CSL component, send the CSLZSHUT FUNC=QUIESCE,SCOPE=CSLMEMBER message to the component you want to shut down.

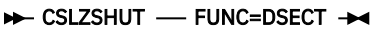
To shut down a CSL and all of its components on a single z/OS image, either:

- Send a CSLZSHUT FUNC=QUIESCE,SCOPE=CSLLOCAL message to the SCI that is active on the z/OS image that contains the CSL to be shut down.
- Send a CSLZSHUT FUNC=QUIESCE,SCOPE=CSLLOCAL,OSNAME=xxxx message to any SCI active in the IMSplex (where xxxx is the z/OS image where the CSL to be shut down is active). SCI sends a CSLZSHUT request to all of the CSL components to be shut down.

To shut down the CSL on an entire IMSplex, send a CSLZSHUT FUNC=QUIESCE,SCOPE=CSLPLEX message to any SCI active in the IMSplex. SCI sends a CSLZSHUT request to all the CSL components in the IMSplex.

CSLZSHUT syntax

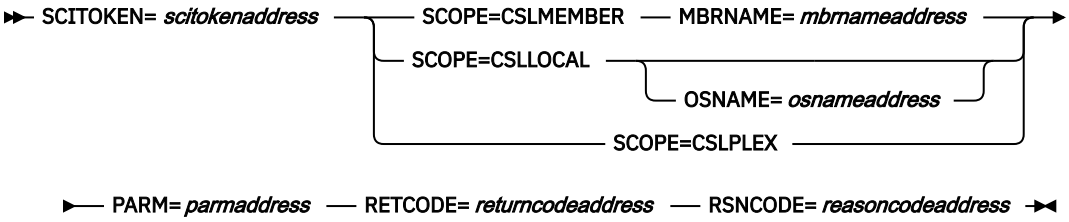
Use FUNC=DSECT to include equate (EQU) statements in your program for the CSLZSHUT parameter list length and the CSLZSHUT return and reason codes.



CSLZSHUT QUIESCE syntax

Use FUNC=QUIESCE to request that a CSL component shut down normally. Any work that the CSL component is currently processing is completed, and then the component shuts down. After processing the request, that component will not accept any new work.

A



If the component that is being shut down is an SCI, the IMSplex members that are currently registered with that SCI are not deregistered before SCI terminates. This can impact event notification. These IMSplex members cannot communicate with other IMSplex members because their SCI is shut down. If

one or more of the "orphaned" members is shut down or fails, the other IMSplex members are not notified of the shutdown or failure event until SCI comes back online.

Notification of the shutdown or failure depends on the authorization level of the members. If the terminating member is non-authorized, other members are notified when SCI restarts. If the terminating member is authorized, other authorized members, including orphaned authorized members, are notified before SCI restarts.

CSLZSHUT parameters

MBRNAME=*symbol MBRNAME=(r2-r12)*
(Required if SCOPE=CSLMEMBER) - Specifies the 8-byte CSL member name to which to send the shutdown request.

OSNAME=*symbol*
OSNAME=*(r2-r12)*
(Optional if SCOPE=CSLLOCAL) - Specifies the 8-byte name of the CSL, running on the z/OS image, that is to be shut down. If the OSNAME parameter is specified and the SCI is not active on the z/OS image specified, the command will not be processed.

PARM=*symbol*
PARM=*(r1-r12)*
(Required) - Specifies the CSLZSHUT parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by ZSHUT_PARMLN.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the return code on output. SCI return codes are defined in CSLSRR. Possible return codes are described in the following table.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. SCI reason codes are defined in CSLSRR. Possible return codes are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

SCOPE=CSLMEMBER | CSLLOCAL | CSLPLEX
(Required) - Specifies the scope of the CSL termination. Valid values for the SCOPE parameter are:

CSLMEMBER
This requests the CSL component receiving the request to shut itself down. CSLMEMBER can be processed by any CSL component.

CSLLOCAL
This requests that the CSL components on a single z/OS image be shut down. If the OSNAME parameter is also specified, the CSL components on that particular z/OS image are shut down. If the OSNAME parameter is specified and the SCI is not active on the z/OS image specified, the command will not be processed. If the OSNAME parameter is not specified, the SCI receiving the message shuts down the CSL on the local z/OS image. Only an SCI can process a SCOPE=CSLLOCAL request. If this request is sent to other CSL components, it is ignored.

CSLPLEX
This requests that the CSL components in an entire IMSplex be shut down. Only an SCI can process a SCOPE=CSLPLEX request. If this request is sent to other CSL components, it is ignored.

CSLZSHUT return and reason codes

The following table lists the return and reason codes that can be returned on a CSLZSHUT macro request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 32. CSLZSHUT return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'xx000008'	X'00002050'	The caller of the service attempted to pass an invalid parameter list. The request is rejected. "xx" identifies the component to set the return code.

Related concepts

“How to interpret CSL request return and reason codes” on page 97

Common Service Layer (CSL) return and reason codes indicate the success or failure of sending the request to the CSL address space and reflect the success or failure of the particular CSL request that is being made.

Chapter 4. CSL automated operator program requests

Certain CSL requests can be used by AOP clients such as TSO SPOC in order to automate some of your operator programs. The following topics describe these requests in detail.

CSLOMCMD: command request

By using the CSLOMCMD request, your AOP client application that is running on the host can issue requests and send commands to OM.

Commands which are submitted through the OM API or REXX SPOC API use the address space identifier (ASID) USERID for authorization.

Commands which are submitted from a program using the OM API while executing in a TSO session use the TSO USERID for authorization.

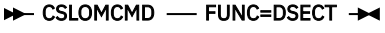
Commands which are submitted from a program using the OM API while executing in a message processing program (MPP) region or batch message processing program (BMP) region use the IMS MPP/BMP dependent region USERID for authorization. In this environment, the actual transaction userid can be used for authorization if the user's installation uses the IMS Build Security Environment exit routine (DFSBSEX0) or OTMA/APPC SECURITY FULL (for example, the user's installation issues the / SECURE OTMA/APPC FULL command).

CSLOMCMD syntax

The syntax for CSLOMCMD can vary depending on what the automated operator client intends to perform.

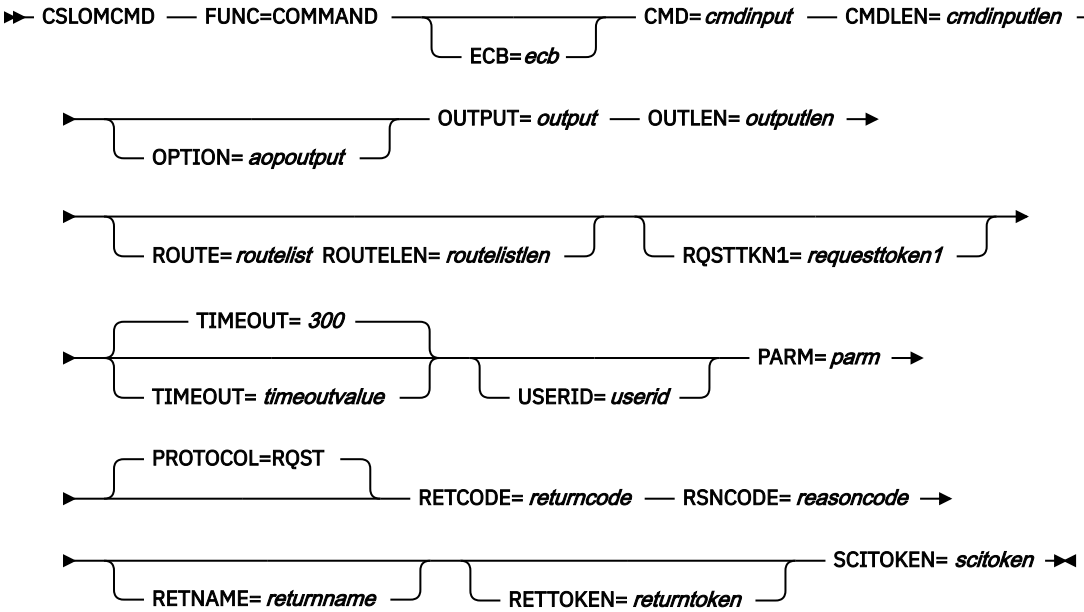
DSECT syntax

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



Request protocol syntax

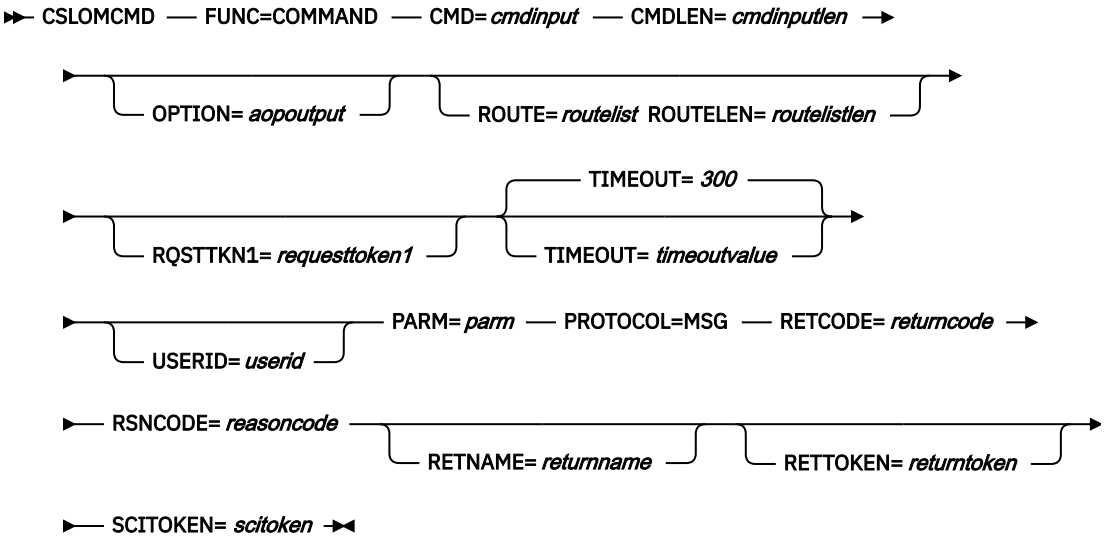
For automation clients that want to wait for the output from the OM request, use this syntax.



The response is passed back to the client after the request is completed.

Message protocol syntax

For automation clients that want to receive command output through their user exit, use this syntax.



The response is passed back to the client using the SCI Input exit. The client must have specified an SCI Input exit (INPUTEXIT=) on the SCI registration request (CSLSCREG) to receive a response.

CSLOMCMD parameters

CMD=*symbol*
CMD=*(r2-r12)*
(Required) - Specifies the command input buffer. This can be any IMS command that can be specified through the OM API. The first character of the command does not need to be a command recognition character (for example, /). The command recognition character does not control command routing in OM. The ROUTE= keyword controls which IMSplex members receive a command. If a command recognition character is entered in the command string it is ignored. The first character in the command is considered a command recognition character if it is not a character between A-Z (either uppercase or lowercase).

CMDLEN=*symbol*
CMDLEN=*(r2-r12)*
(Required) - Specifies the length of the command input buffer.

ECB=*symbol*
ECB=*(r2-r12)*
(Optional) - Specifies the address of a z/OS event control block (ECB) used for asynchronous requests. When the request is complete, the ECB specified is posted. If an ECB is not specified, the task is suspended until the request is complete. If an ECB is specified, the invoker of the macro must issue a WAIT (or equivalent) after receiving control from CSLOMCMD before using or examining any data returned by this macro (including the RETCODE and RSNCODE fields).

OPTION=*aopoutput*
OPTION=*(r2-r12)*
(Optional) - Use OPTION to return the format identifiers (FID) in the output from command processing clients. For example, when a type-1 /DISPLAY command is sent to an IMS command processing client, you can request that the FID be returned in each output line. The FID indicates to an AOI program how to map the line of output. The FID can be useful if you are converting existing AOI programs to OM AOI programs.

If OPTION is specified as a register, the register must contain the option value. For example, the value of AOPOUTPUT is 1. Therefore, the register must contain a 1.

The CSLOMCMD request contains the equate for the value of AOPOUTPUT. The DSECTS for the output of CSLOMCMD when OPTION=AOPOUTPUT are described in the DISPLAY macro in the IMS.SDFSMAC data set.

OUTLEN=*symbol*
OUTLEN=*(r2-r12)*
(Required for RQST) - Specifies a 4-byte field to receive the length of the output returned by the CSLOMCMD request. OUTLEN contains the length of the output pointed to by the OUTPUT= parameter.

The output length is zero if no output is built, for example, if an error is detected before any output can be built.

OUTPUT=*output*
OUTPUT=*(r2-r12)*

(Required for RQST) - Specifies a 4-byte field to receive the address of the variable length output returned by the CSLOMCMD request. The output contains the command response output. The output length is returned in the OUTLEN= field.

The output address is zero if no output was built, for example, if an error was detected before any output could be built.

The output buffer is not preallocated by the caller. After the request returns it, this word contains the address of a buffer containing the update output. It is the caller's responsibility to release this storage by issuing the CSLSCBFR FUNC=RELEASE request when it is finished with the storage. The length of the output is returned in the OUTLEN= field.

PARM=*symbol*
PARM=*(r1-r12)*
(Required) - Specifies the CSLOMCMD parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by OCMD_PARMLN.

PROTOCOL=RQST
PROTOCOL=MSG
(Optional) - Specifies the SCI protocol for sending the request to OM.

- RQST - Send command to OM using the SCI request protocol.
- MSG - Send command to OM using the SCI message protocol.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the return code on output. OM return codes are defined in the CSLORR. SCI return codes are defined in CSLSRR.

The return code can be from OM (CSLOMCMD) or SCI (CSLSCMSG or CSLSCRQS). If ECB is specified, the RETCODE is not valid until the ECB is posted. All return codes contain the SCI member type indicator for either SCI, OM, or RM in the high order byte (X'01' for SCI, X'02' for OM, X'03' for RM).

RETNAME=*symbol*
RETNAME=*(r2-r12)*
(Optional) - Specifies an 8-byte output field to receive the OM name. This is the CSL member name of the target address space to which SCI sent the request.

RETTOKEN=*symbol*
RETTOKEN=*(r2-r12)*
(Optional) - Specifies a 16-byte output field to receive the OM SCI token returned to the caller. This is the OM SCI token for the target address space to which the request was sent.

ROUTE=*symbol*
ROUTE=*(r2-r12)*
(Optional) - Specifies a route list that identifies OM clients (for example, IMS control regions) in the IMSplex to which the command is sent. If you do not specify ROUTE, OM routes to all clients that are registered and ready to process commands. If the route list specified consists of a SYSID of an OM client that is not registered for the specified command, then the command will fail with return and reason codes indicating the client is not registered for the command. For example, if a QUERY

IMSPLEX command with ROUTE=IMS1 is processed by the OM address space and the IMS control region IMS1 is not registered for this command, then the command fails.

- To explicitly route the command to all command processing clients that have registered for and are ready to process commands, specify ROUTE=*.
- To route the command to the first command processing client which is READY and has MASTER capability, specify ROUTE=%. With ROUTE=%, OM routes the command to only one command processing client that OM chooses.

Note: Use commas to separate a list of client names.

ROUTELEN=*symbol*
ROUTELEN=*(r2-r12)*
(Optional) - Specifies the length of the list specified in the ROUTE= parameter.

RQSTTKN1=*symbol*
RQSTTKN1=*(r2-r12)*
(Optional) - Specifies a 16-byte user generated request token that is used to associate the request response with the request for asynchronous processing. RQSTTKN1 can include A-Z, 0-9, or printable characters (not case sensitive), except &, <, and >. OM returns the request token encapsulated in the <rqsttkn1></rqsttkn1> tags in the XML output. OM converts any invalid data to periods (.) before returning XML output to the client. For PROTOCOL=MSG requests, OM also returns the address of this token in the OM Directive parameter list (mapped by CSLOMDIR macro) in the field ODIR_CQRT1PTR. This parameter must be 16 bytes and, if necessary, padded with blanks.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. OM reason codes are defined in the CSLORR. SCI reason codes are defined in CSLSRR.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token is returned by a successful `CSLSCREG FUNC=REGISTER` request.

TIMEOUT=*timeoutvalue*
TIMEOUT=*symbol*
TIMEOUT=*(r2-r12)*
(Optional) - Specifies a 4-byte command timeout value in seconds. If the TIMEOUT value is reached during OM command processing and before all clients have responded to the command, OM terminates the command and returns all available responses. If too small a value is specified, an incomplete response is returned. The TIMEOUT value ensures a response is returned even if a client processing the command is unable to respond. The TIMEOUT keyword is ignored if no CMD keyword is specified. If a command is requested but no timeout value is specified, a timeout value of 5 minutes is used.

If TIMEOUT is specified as a symbol, the symbol must be an EQU symbol equated to the timeout value. If TIMEOUT is specified as a number, the number must be the timeout value.

USERID=*symbol*
USERID=*(r2-r12)*
(Optional) - Specifies the 8-byte user ID to be used by RACF® or an equivalent security product. Use this parameter only if your client address space has been authorized for this request. If your client is unauthorized, the user ID is obtained automatically from z/OS control blocks. This user ID is intended for use by authorized system management address spaces that can issue an OM request on behalf of another address space or remote client. In this case, the user ID of the client address space is not the user ID of the actual client, so it must be passed to OM. This parameter must be 8 bytes, left-aligned, and, if necessary, padded with blanks.

You can find the return and reason codes for the CSLOMCMD command request in CSLOMCMD return and reason codes (Messages and Codes).

Related reference

"CSLSCREG: registration request" on page 215
The Structured Call Interface (SCI) registration request is used to create a connection between an IMSplex member and SCI. Before SCI can be used for communication within the IMSplex, an IMSplex member must issue the CSLSCREG request and receive an SCI token when the request completes in order to be recognized by SCI.

"CSL Operations Manager XML output" on page 233
Command responses that are returned through the OM API are embedded in XML tags using codepage 037. XML output is generated for responses to the CSLOMI, CSLOMCMD, and CSLOMQRY requests.

CSLOMI: API request

With the CSLOMI request, your AOP client can communicate with a z/OS address space that acts as an OM AOP client. You can then issue OM requests and send QUERY commands to OM.

With the CSLOMI request, a z/OS automated operator client can issue an IMS command to or request OM-specific information from an OM. The CSLOMI macro interface is designed for use by system management address spaces that receive input from a workstation or other z/OS address space and must pass the request to OM. In this case the workstation application builds the input string and passes it to the z/OS address space. The z/OS address space passes the input string to OM on the INPUT= parameter.

Commands which are submitted through the OM API or REXX SPOC API use the address space identifier (ASID) USERID for authorization.

Commands which are submitted from a program using the OM API while executing in a TSO session use the TSO USERID for authorization.

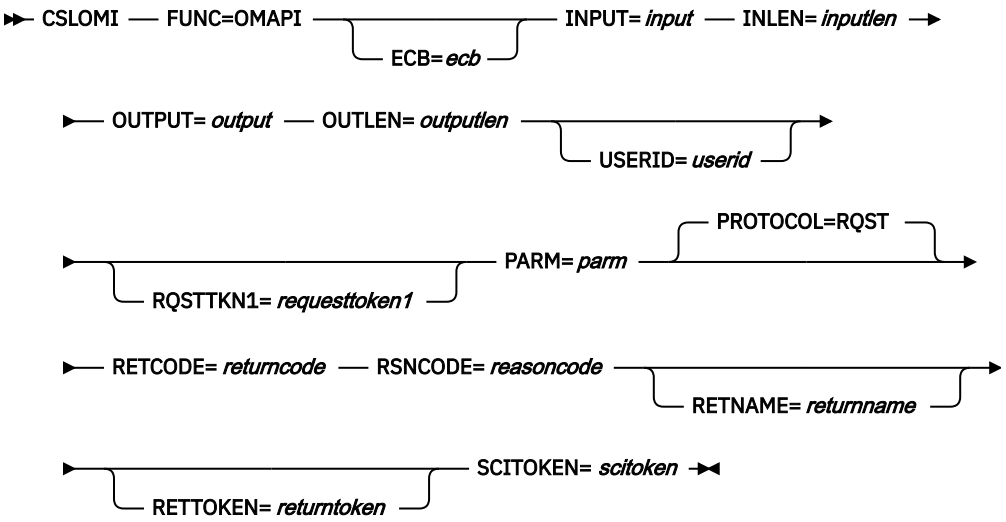
Commands which are submitted from a program using the OM API while executing in a message processing program (MPP) region or batch message processing program (BMP) region use the IMS MPP/BMP dependent region USERID for authorization. In this environment, the actual transaction userid can be used for authorization if the user's installation uses the IMS Build Security Environment exit routine (DFSBSEX0) or OTMA/APPC SECURITY FULL (for example, the user's installation issues the / SECURE OTMA/APPC FULL command).

CSLOMI syntax

The syntax for CSLOMI can vary, depending on how the automated operator client wants to receive the command response. If the client does not have an input exit and wants to receive the command output as a response, use the request syntax. If the client does have an input exit and wants to receive the command output as a message, use the message syntax.

CSLOMI request protocol syntax

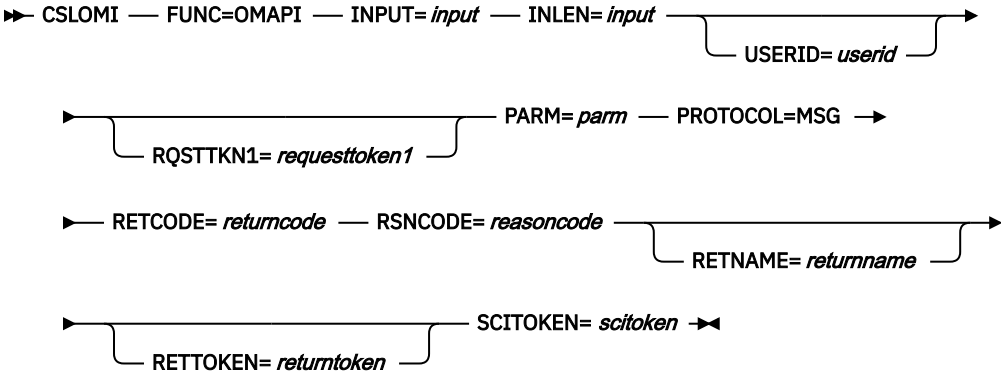
For automated clients that want to wait for output from the OM request, use this syntax.



After control is returned to the client (if ECB is not specified), or the ECB is posted (if an ECB is specified), the response is available to the client.

CSLOMI message protocol syntax

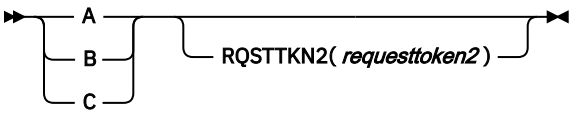
For automated clients that want to receive command output through their user exit, use this syntax:



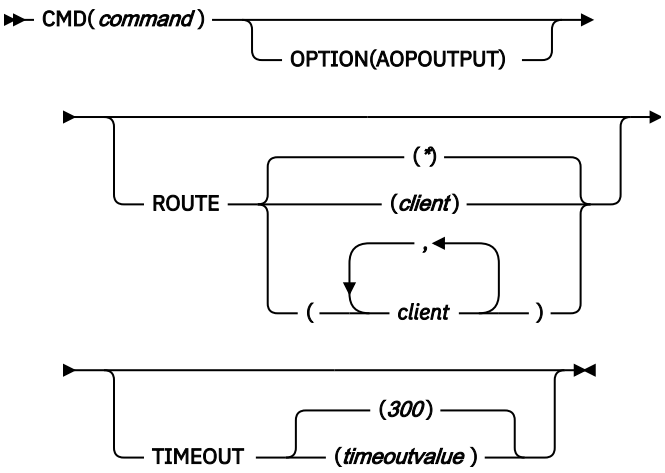
The response is passed back to the client using the SCI Input exit. The client must have specified an SCI Input exit (INPUTEXIT=) on the SCI registration request (CSLSCREG) to receive a response.

CSLOMI Input= parameter syntax

For other applications or workstations that do not communicate directly with OM, use this syntax.

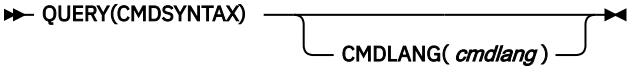


A



B

C



This syntax is used for the INPUT= parameter. The application builds the command or query, and passes it to a z/OS address space that communicates with OM directly.

CSLOMI request and message parameters

ECB=*symbol*
ECB=*(r2-r12)*
(Optional) - Specifies the address of a z/OS event control block (ECB) used for asynchronous requests. When the request is complete, the ECB specified is posted. If an ECB is not specified, the task is suspended until the request is complete. If an ECB is specified, the invoker of the macro must issue a WAIT (or equivalent) after receiving control from CSLOMI before using or examining any data returned by this macro (including the RETCODE and RSNCODE fields).

INLEN=*symbol*
INLEN=*(r2-r12)*
(Required) - Specifies the length of the input buffer.

INPUT=*symbol*
INPUT=*(r2-r12)*
(Required) - Specifies the address of the input buffer.

The following shows an example of the input buffer that is passed to CSLOMI. The input buffer is the character field MYINPUT and specifies three parameters: a command string of QRY TRAN SHOW(ALL), a timeout value of 360 seconds, and a route list consisting of one element, IMSA:

```
CSLOMI FUNC=OMAPI,INPUT=MYINPUT,INLEN=INPUTLEN
INPUTLEN DC    A(MYINPUTL)
MYINPUT  DC    C'CMD (QRY TRAN SHOW(ALL) TIMEOUT(360) ROUTE(IMSA)'
MYINPUTL EQU   *-MYINPUT
```

OUTLEN=*symbol*
OUTLEN=*(r2-r12)*
(Required for RQST) - Specifies a 4-byte field to receive the length of the output returned by the CSLOMI request. OUTLEN contains the length of the output pointed to by the OUTPUT= parameter.

The output length is zero if no output is built, for example, if an error is detected before any output can be built.

OUTPUT=*symbol*
OUTPUT=*(r2-r12)*
(Required) - Specifies a field to receive the variable length output returned by the CSLOMI request. The output contains the command response output. The output length is returned in the OUTLEN= field.

The output address is zero if no output was built, for example, if an error was detected before any output could be built.

The output buffer is not preallocated by the caller. After the request returns it, this word contains the address of a buffer containing the update output. It is the caller's responsibility to release this storage by issuing the CSLSCBFR FUNC=RELEASE request when it is finished with the storage. The length of the output is returned in the OUTLEN= field.

PARM=*symbol*
PARM=*(r2-r12)*
(Required) - Specifies the CSLOMI parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by OI_PARMLN.

PROTOCOL=RQST
PROTOCOL=MSG
(Optional) - Specifies the SCI protocol for sending the request to OM.

- RQST - Send command to OM using the SCI request protocol.
- MSG - Send command to OM using the SCI message protocol.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the return code on output. OM return codes are defined in the CSLORR. SCI return codes are defined in CSLSRR.

The return code can be from OM (CSLOMI) or SCI (CSLSCMSG or CSLSCRQS). If ECB is specified, the RETCODE is not valid until the ECB is posted. All return codes contain the SCI member type indicator for either SCI, OM, or RM in the high order byte (X'01' for SCI, X'02' for OM, X'03' for RM).

RETNAME=*symbol*
RETNAME=*(r2-r12)*
(Optional) - Specifies an 8-byte output field to receive the OM name. This is the CSL member name of the target address space to which SCI sent the request.

RETTOKEN=*symbol*
RETTOKEN=*(r2-r12)*
(Optional) - Specifies a 16-byte output field to receive the OM SCI token returned to the caller. This is the OM SCI token for the target address space to which the request was sent.

RQSTTKN1=*symbol*
RQSTTKN1=*(r2-r12)*
(Optional) - Specifies a 16-byte user generated request token that is used to associate the request response with the request for asynchronous processing. RQSTTKN1 can include A-Z, 0-9, or printable characters (not case sensitive), except &, <, and >. OM returns the request token encapsulated in the <rqsttkn1></rqsttkn1> tags in the XML output. OM converts any invalid data to periods (.) before returning XML output to the client. For PROTOCOL=MSG requests, OM also returns the address of this token in the OM Directive parameter list (mapped by CSLOMDIR macro) in the field ODIR_CQRT1PTR. This parameter must be 16 bytes and, if necessary, padded with blanks.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. OM reason codes are defined in the CSLORR. SCI reason codes are defined in CSLSRR. Possible reason codes are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token is returned by a successful CSLSCREG FUNC=REGISTER request.

USERID=*symbol*
USERID=*(r2-r12)*
(Optional) - Specifies the 8-byte user ID that should be used for security checking for the command and keyword combination. This user ID is used only if the client address space is an authorized caller. If the client address space is unauthorized, the user ID is obtained from z/OS control blocks. This user ID is intended for use by authorized system management address spaces that can issue an OM request on behalf of another address space or remote client. In this case, the user ID of the client address space is not the user ID of the actual client, so it must be passed to OM. This parameter must be 8 bytes, left-justified, and, if necessary, padded with blanks.

CSLOMI Input= parameters

The parameters for the CSLOMI input option are for applications and workstations that do not communicate directly with OM.

CMD(*command*)
(Required if QUERY is not specified) - Specifies the command input buffer. This can be any IMS command that can be specified through the OM API. The first character of the command does not need to be a command recognition character (for example, /). The command recognition character does not control command routing in OM. The ROUTE keyword is used to control which IMSplex members receive a command. If a command recognition character is entered in the command string, it is ignored. The first character in the command is considered a command recognition character if it is not a character between A-Z (either uppercase or lowercase).

CMDLANG(*cmdlang*)
The language to be used for IMS command text that is returned on the request. This value defaults to the default established for the OM system specified on the OM startup parameter CMDLANG=. Currently the only accepted value is ENU for US English. If an invalid language is specified text in the OM default language is returned.

OPTION(AOPOUTPUT)
(Optional, valid only for CMD()) - Specify the AOPOUTPUT option to return the format identifiers (FID) in the output from command processing clients. For example, when a type-1 /DISPLAY command is sent to an IMS command processing client, you can request that the FID be returned in each output line. The FID indicates to an AOI program how to map the line of output. The FID can be useful if you are converting existing AOI programs to OM AOI programs.

QUERY(*querytype*)
Type of query to be performed by OM.

CMDCLIENTS
Requests that OM return a list of all clients (for example, IMS control regions) that have registered to OM for command processing.

The list of clients is returned encapsulated in <cmdclients> </cmdclients> tags. *querytype* can be one of the following.

- <mbr name=*membername*>

  The member name is the name of the client address space.

  – <typ> </typ>

    The member type is the type of the client address space.

  – <styp> </styp>

    The member subtype is the subtype of the client address space.

  – <vsn> </vsn>

The member version is the version of the client address space.

– <jobname> </jobname>

The client jobname is the jobname or the started task for the client address space.

• </mbr>

CMDSYNTAX
Requests that OM return a list of the XML representing the command syntax for selected commands registered with OM. Additionally, the translatable text associated with the command syntax is returned.

The command syntax XML is returned encapsulated in <cmdsyntax> </cmdsyntax> tags. The command syntax DTD is returned encapsulated in <cmddtd> </cmddtd> tags. The command syntax translatable text is returned encapsulated in <cmdtext> </cmdtext> tags.

ROUTE(*routelist*)
(Optional) - Specifies a route list that identifies OM clients (for example, IMS control regions) in the IMSplex to which the command is sent. In the list, the clients are separated by commas. If you do not specify ROUTE, OM routes to all clients that are registered and ready to process commands.

- To explicitly route the command to all command processing clients that have registered for and are ready to process commands, specify ROUTE(*).
- To route the command to the first command processing client which is READY and has MASTER capability, specify ROUTE(%). With ROUTE(%), OM routes the command to only one command processing client that OM chooses.

RQSTTKN2(*requesttoken2*)
(Optional) - Specifies a 16-byte user generated request token that is used to associate the request response with the request for asynchronous processing. RQSTTKN2 can include A-Z, 0-9, or printable characters (not case sensitive), except &, <, and >. OM returns the request token encapsulated in the <rqsttkn2></rqsttkn2> tags in the XML output. OM converts any invalid data to periods (.) before returning XML output to the originating client. For PROTOCOL=MSG requests, OM also returns the address of this token in the OM Directive parameter list (mapped by CSLOMDIR macro) in the field ODIR_CQRT2PTR.

TIMEOUT(*timeoutvalue*)
(Optional) - Specifies a 4-byte command timeout value in seconds. If the TIMEOUT value is reached during OM command processing before all clients have responded to the command, OM terminates the command and returns all available responses. If too small a value is specified, an incomplete response is returned. The TIMEOUT value ensures a response is returned even if a client processing the command cannot respond. The TIMEOUT keyword is ignored if no CMD keyword is specified. If a command is requested but no timeout value is specified, a timeout value of 5 minutes is used.

CSLOMI return and reason codes

The following table lists the return and reason codes that can be returned on a CSLOMI macro request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 33. CSLOMI return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'02000004'	Any code	This return code represents a warning. All or part of the request might have completed successfully. Additional information is returned with the response to the request.

Table 33. CSLOMI return and reason codes (continued)

Return code	Reason code	Meaning
	X'00001000'	The specified command timed out before all of the command response information could be collected. One or more clients might not be responding, or a client might have needed more time to process the command. If a TIMEOUT value is specified, ensure the value is long enough to allow for the command to be processed. All command response information that is collected prior to the timeout is returned.
	X'00001004'	The INPUT exit rejected the command contained in the CMD field. The command was not processed.
	X'00001008'	The client (specified in the corresponding XML <mbr></mbr> tags in the <cmderr> section) was specified in the ROUTE list for the command specified in the CMD field. The command was not routed to the command processing client because the client is not the master.
	X'00001010'	The text file could not be loaded in the language specified on the CMDLANG parameter. The default language is used.
	X'00001014'	The command completed with warnings. Check the return codes. At least one client member returned a return code 4 to the Operations Manager. All other clients returned a return code not greater than 4. If the command was successfully processed by one command processing client as designed, but all other command processing clients to which the command was routed received reason code X'00001000' (IRSN_NOTMSTR) for return code X'00000004', then the overall OM return and reason codes will be: • WARNING (X'02000004') • WARNING (X'00001014') Refer to the completion codes returned on the request for further information.
X'02000008'	Any code	This return code represents a parameter error. The request was not processed due to the error.
	X'00002000'	The command specified in the CMD field is invalid.
	X'00002004'	The primary keyword specified in the CMD field is invalid with the command specified.
	X'00002028'	An invalid keyword was specified in the CMD field.
	X'0000202C'	BPE detected an unknown positional parameter in the command in the CMD field.
	X'00002030'	A keyword was specified with an equal sign (KEYWORD=) when a sublist was expected (KEYWORD()) in the command in the CMD field.
	X'00002034'	An incomplete keyword or keyword parameter was specified in the command in the CMD field.
	X'00002038'	A keyword is missing from the command in the CMD field.

Table 33. CSLOMI return and reason codes (continued)

Return code	Reason code	Meaning
	X'0000203C'	The value of a keyword parameter specified in the command was invalid.
	X'00002040'	A duplicate keyword was specified in the command in the CMD field.
	X'00002044'	Text containing the syntax error is returned in the XML <message></message> tags.
	X'00002048'	More than one filter was specified.
	X'00002050'	The caller of the service attempted to pass an invalid parameter list. The request is rejected.
X'0200000C'	Any code	This return code represents a list error. The request might or might not have been processed due to the error. Refer to the XML tag <cmderr> section and the completion codes for each command processing client listed in the XML tag <cmdrspdata> section.
	X'00003000'	The command was routed to multiple clients. At least one client was able to process the request successfully and return either command response data or a response message. Refer to the completion codes returned on the request for further information.
	X'00003004'	The command was routed to multiple clients. None of the clients was able to process the request successfully. No command response data or response messages were returned by any client.
	X'00003008'	The command was routed to multiple clients. None of the clients that processed the command returned a return code 0 and reason code 0 to OM. At least one command client returned either command response data or a response message.
	X'0000300C'	The command was routed to multiple clients. Not all of the clients that processed the command returned a return code 0 and reason code 0 to the OM. Also, at least one client returned a return code 4. Refer to the completion codes returned on the request for additional information.
X'02000010'	Any code	This return code represents an environmental error. The request could not be processed at this time due to the current environment. This condition might be temporary.
	X'00004000'	The command contained in the CMD field could not be processed by the client indicated in the corresponding XML <mbr></mbr> tags in the <cmderr> section because the client was not yet ready to process commands.
	X'00004004'	The command contained in the CMD field could not be processed by the client indicated in the corresponding XML <mbr></mbr> tags in the <cmderr> section because the client was not registered for the command.

Table 33. CSLOMI return and reason codes (continued)

Return code	Reason code	Meaning
	X'00004008'	The command contained in the CMD field could not be processed by the client indicated in the corresponding XML <mbr></mbr> tags in the <cmderr> section because the client is not active in the IMSplex.
	X'0000400C'	The command contained in the CMD field could not be processed by the client indicated in the corresponding XML <mbr></mbr> tags in the <cmderr> section because the client registered for the command with invalid PADEF grammar.
	X'00004010'	The command contained in the CMD field could not be processed. The client that issued the command is not authorized. Examine the <cmdsecerr> section in the XML file to determine why the client is not authorized.
	X'00004014'	A data set allocation error occurred; the data set specified by the CMDTEXT= DSN parameter could not be allocated.
	X'00004018'	A data set read error occurred; a member in the data set specified by the CMDTEXT= DSN could not be read. The member name is CSLOT concatenated with the 3-character CMDLANG value.
	X'00004020'	The parameter list version is invalid.
X'02000014'	Any code	This return code represents a system error. An internal error occurred, and the command was not processed.
	X'00005000'	An OM internal error occurred. Due to a storage shortage, OM was unable to allocate a CMD block to process the command in the CMD field.
	X'00005004'	An OM internal error occurred. Due to a storage shortage, OM was unable to allocate a CRSP block to process the command in the CMD field.
	X'00005008'	An OM internal error occurred. Due to a storage shortage, OM was unable to allocate the command input buffer to process the command in the CMD field.
	X'0000500C'	An OM internal error occurred. OM was unable to obtain the VERB latch while processing the command in the CMD field.
	X'00005010'	An OM internal error occurred. Due to a storage shortage, OM was unable to obtain storage for the parsed output blocks to parse the command in the CMD field.
	X'00005014'	An OM internal error occurred. OM was unable to add the CMD block to the command instance hash table while processing the command in the CMD field.
	X'00005018'	An OM internal error occurred. OM was unable to find the CMD block in the command instance hash table while processing the command in the CMD field.
	X'0000501C'	An OM internal error occurred. OM was unable to scan for the CMD block in the command instance hash table while processing the command in the CMD field.

Table 33. CSLOMI return and reason codes (continued)

Return code	Reason code	Meaning
	X'00005020'	An OM internal error occurred. OM was unable to obtain a system AWE while processing the command in the CMD field. The command was not processed by the command processing client. Refer to the <cmderr> section in the XML file for the member name of the command processing client.
	X'00005024'	An OM internal error occurred. OM was unable to queue a system AWE while processing the commanding the CMD field. The command was not processed by the command processing client. Refer to the <cmderr> section of the XML file for the member name of the command processing client.
	X'00005028'	An OM internal error occurred. OM was unable to parse the command contained in the CMD field due to a BPEPARSE internal error.
	X'0000502C'	An OM internal error occurred. The command output header allocation failed.
	X'00005030'	An OM internal error occurred. The command output response allocation failed.
	X'00005034'	An OM internal error occurred. The OUTPUT buffer allocation failed.
	X'00005038'	An OM internal error occurred. The VERB hash table add failed.
	X'0000503C'	An OM internal error occurred. The CLNT block could not be obtained.
	X'00005040'	An OM internal error occurred. The CSLSCQRY request failed.
	X'00005044'	An OM internal error occurred. OM could not obtain storage to pass a copy of the command grammar to the BPEPARSE service.

Related reference

"CSLSCREG: registration request" on page 215
The Structured Call Interface (SCI) registration request is used to create a connection between an IMSplex member and SCI. Before SCI can be used for communication within the IMSplex, an IMSplex member must issue the CSLSCREG request and receive an SCI token when the request completes in order to be recognized by SCI.

"CSL Operations Manager XML output" on page 233
Command responses that are returned through the OM API are embedded in XML tags using codepage 037. XML output is generated for responses to the CSLOMI, CSLOMCMD, and CSLOMQRY requests.

"CSL OM directives" on page 159

An OM directive is a function that OM defines that can be sent as a message to OM clients to inform the OM clients of work to be processed. Any command processing client that has registered commands to OM can be selected to perform an OM directive.

CSLOMQRY: query request

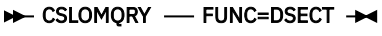
With the CSLOMQRY request, any AOP client that is running on the host can request OM-specific information.

CSLOMQRY syntax

The syntax for CSLOMQRY can vary depending on what the automated operator client intends to perform. Parameter descriptions for each syntax example are provided in the following section.

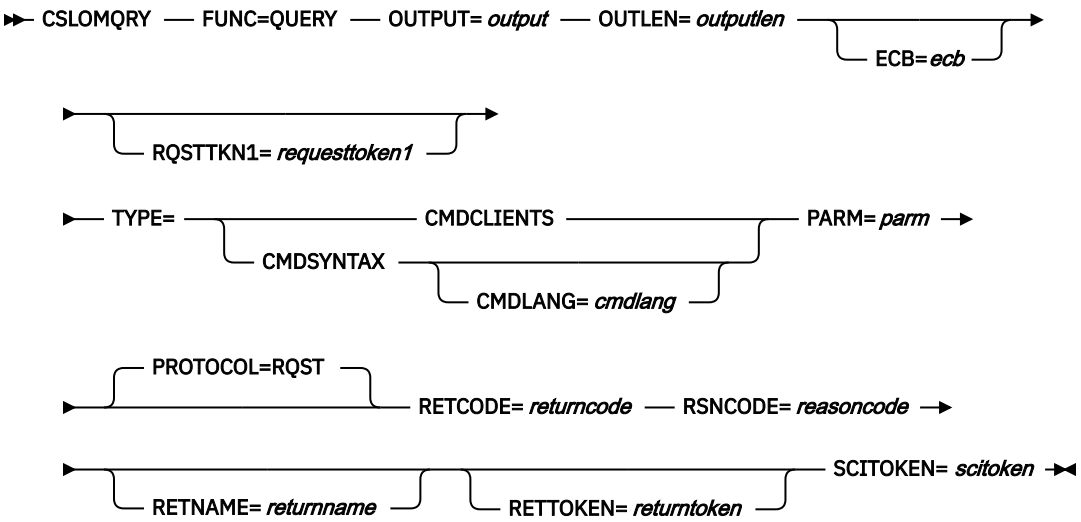
DSECT syntax

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



Request protocol syntax

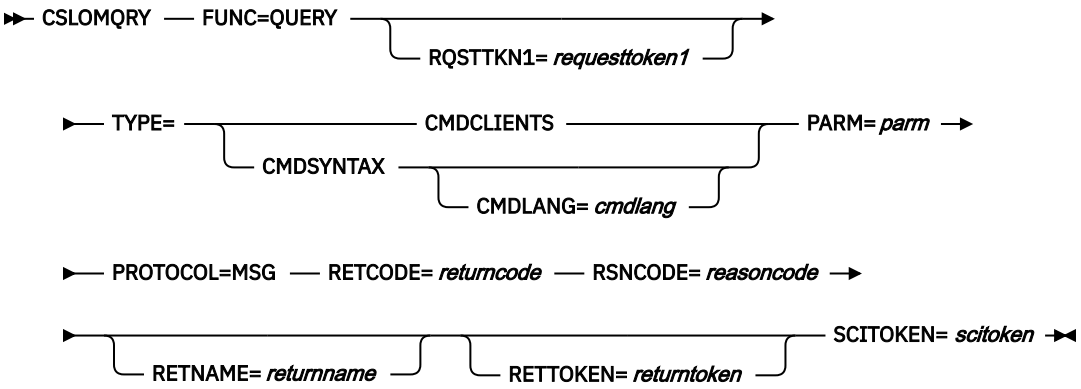
For automation clients that want to wait for the output from the OM request, use this syntax.



The response is passed back to the client after the request is completed.

Message protocol syntax

For automation clients that want to send a message to OM to process an OM request, use this syntax.



The response is passed back to the client using the SCI Input exit. The client must have specified an SCI Input exit (INPUTEXIT=) on the SCI registration request (CSLSCREG) to receive a response.

CSLOMQRY parameters

CMDLANG=*cmdlang*
(Optional) - The language to be used for IMS command text that is returned on the request. This value defaults to the default established for the OM system specified on the OM startup parameter CMDLANG=. Currently the only accepted value is ENU for US English. If an invalid language is specified in OM, the default language is returned.

ECB=*symbol*
ECB=*(r2-r12)*
(Optional) - Specifies the address of a z/OS event control block (ECB) used for asynchronous requests. When the request is complete, the ECB specified is posted. If an ECB is not specified, the task is suspended until the request is complete. If an ECB is specified, the invoker of the macro must issue a WAIT (or equivalent) after receiving control from CSLOMQRY before using or examining any data returned by this macro (including the RETCODE and RSNCODE fields).

OUTLEN=*symbol*
OUTLEN=*(r2-r12)*
(Required for RQST) - Specifies a 4-byte field to receive the length of the output returned by the CSLOMQRY request. OUTLEN contains the length of the output pointed to by the OUTPUT= parameter.

The output length is zero if no output is built, for example, if an error is detected before any output can be built.

OUTPUT=*output*
OUTPUT=*(r2-r12)*
(Required) - Specifies a field to receive the variable length output returned by the CSLOMQRY request. The output contains the command response output. The output length is returned in the OUTLEN= field.

The output address is zero if no output was built, for example, if an error was detected before any output could be built.

The output buffer is not preallocated by the caller. After the request returns it, this word contains the address of a buffer containing the update output. It is the caller's responsibility to release this storage by issuing the CSLSCBFR FUNC=RELEASE request when it is finished with the storage. The length of the output is returned in the OUTLEN= field.

PARM=*symbol*
PARM=*(r2-r12)*
(Required) - Four-byte input parameter that specifies the address of the storage used by the request to pass the parameters to SCI. The length of the parameter list must be equal to the parameter list length EQU value defined by OQRY_PARMLN.

PROTOCOL=RQST
PROTOCOL=MSG

(Optional) - Specifies the SCI protocol for sending the request to OM.

- RQST - Send command to OM using the SCI request protocol.
- MSG - Send command to OM using the SCI message protocol.

RETCODE=*symbol*
RETCODE=*(r2-r12)*

(Required) - Specifies a 4-byte field to receive the return code on output. OM return codes are defined in the CSLORR. SCI return codes are defined in CSLSRR.

The return code can be from OM (CSLOMQRY) or SCI (CSLSCMSG or CSLSCRQS). If ECB is specified, the RETCODE is not valid until the ECB is posted. All return codes contain the SCI member type indicator for either SCI, OM, or RM in the high order byte (X'01' for SCI, X'02' for OM, X'03' for RM).

RETNAME=*symbol*
RETNAME=*(r2-r12)*

(Optional) - Specifies an 8-byte output field to receive the OM name. This is the CSL member name of the target address space to which SCI sent the request.

RETTOKEN=*symbol*
RETTOKEN=*(r2-r12)*

(Optional) - Specifies a 16-byte output field to receive the OM SCI token returned to the caller. This is the OM SCI token for the target address space to which the request was sent.

RQSTTKN1=*symbol*
RQSTTKN1=*(r2-r12)*

(Optional) - Specifies a 16-byte user generated request token that is used to associate the request response with the request for asynchronous processing. RQSTTKN1 can include A-Z, 0-9, or printable characters (not case sensitive), except &, <, and >. OM returns the request token encapsulated in the <rqsttkn1></rqsttkn1> tags in the XML output. OM converts any invalid data to periods (.) before returning XML output to the client. For PROTOCOL=MSG requests, OM also returns the address of this token in the OM directive parameter list (mapped by CSLOMDIR macro) in the field ODIR_CQRT1PTR. This parameter must be 16 bytes and, if necessary, padded with blanks.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*

(Required) - Specifies a 4-byte field to receive the reason code on output. OM reason codes are defined in the CSLORR. SCI reason codes are defined in CSLSRR. Possible reason codes are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*

(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token is returned by a successful `CSLSCREG FUNC=REGISTER` request.

TYPE=CMDCLIENTS
TYPE=CMDSYNTAX

(Required) - Four-byte input parameter that specifies the type of query to be performed by OM.

CMDCLIENTS

Requests that OM return a list of all clients (for example, IMS control regions) that have registered to OM for command processing.

The clients are returned encapsulated in <cmdclients> </cmdclients> tags.

- <mbr name=*membername*>

  The member name is the name of the client address space.

  – <typ> </typ>

    The member type is the type of the client address space.

  – <styp> </styp>

The member subtype is the subtype of the client address space.

– <vsn> </vsn>

The member version is the version of the client address space.

– <jobname> </jobname>

The client jobname is the jobname or the started task for the client address space.

- </mbr>

CMDSYNTAX

Requests that OM return a list of the XML representing the command syntax for selected commands registered with OM. Additionally, the translatable text associated with the command syntax is returned.

The command syntax XML is returned encapsulated in <cmdsyntax> </cmdsyntax> tags. The command syntax DTD is returned encapsulated in <cmddtd> </cmddtd> tags. The command syntax translatable text is returned encapsulated in <cmdtext> </cmdtext> tags.

The command syntax and translatable text that is returned as a result of the CSLOMQRY QUERY TYPE(CMDSYNTAX) request includes information for type-2 commands.

CSLOMQRY return and reason codes

The following table lists the return and reason code combinations that can be returned on a CSLOMQRY request and that are unique to the CSLOMQRY request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 34. CLSOMQRY return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'02000004'	Any code	This return code represents a warning. All or part of the request might have completed successfully. Additional information is returned with the response to the request.
	X'00001010'	The text file could not be loaded in the language specified on the CMDLANG parameter. The default language is used.
X'02000008'	X'00002050'	The caller of the service attempted to pass an invalid parameter list. The request is rejected.
X'02000010'	Any code	This return code represents an environmental error. The request could not be processed at this time due to the current environment. This condition might be temporary.
	X'00004014'	A data set allocation error occurred; the data set specified by the CMDTEXTDSN= parameter in the OM Initialization PROCLIB member (CSLOIxxx) could not be allocated.
	X'00004018'	A data set read error occurred; a member in the data set specified by the CMDTEXTDSN= parameter in the OM Initialization PROCLIB member (CSLOIxxx) could not be read. The member name is 'CSLOT' concatenated with the 3-character CMDLANG value.
	X'00004020'	The parameter list version is invalid.

Related reference

“CSLSCREG: registration request” on page 215

The Structured Call Interface (SCI) registration request is used to create a connection between an IMSplex member and SCI. Before SCI can be used for communication within the IMSplex, an IMSplex

member must issue the CSLSCREG request and receive an SCI token when the request completes in order to be recognized by SCI.

"CSL Operations Manager XML output" on page 233
Command responses that are returned through the OM API are embedded in XML tags using codepage 037. XML output is generated for responses to the CSLOMI, CSLOMCMD, and CSLOMQRY requests.

"CSL OM directives" on page 159
An OM directive is a function that OM defines that can be sent as a message to OM clients to inform the OM clients of work to be processed. Any command processing client that has registered commands to OM can be selected to perform an OM directive.

"CSLOMQRY output" on page 237
Each of the command syntax examples contain a sample of CSLOMQRY XML output. The examples present different scenarios that generate XML output based on the commands that are used in the example.

CSL OM automated operator program clients

OM provides an API interface for application programs that automate operator actions known as automated operator programs (AOP). You can use an AOP to issue commands that are embedded in an OM API request to an OM.

OM provides an application programming interface (API) for application programs that automate operator actions. These programs are called automated operator programs (AOP). An AOP issues commands that are embedded in an OM API request to an OM. The responses to those commands are returned to the AOP embedded in XML tags.

If you want to use OM to manage commands and command responses in an IMSplex for your own product or service, you can use an AOP client, such as each of these clients:

- The IMS-supplied AOP client, TSO single point of control (SPOC), which runs on the host. With the TSO SPOC, an automated operator can issue commands to the IMSplex and receive responses to those commands interactively.
- An AOP client that runs on a workstation (called a workstation SPOC).
- A command processing client, such as IMS.

An OM client uses OM requests to communicate with OM. Each OM client must register to SCI before it can issue OM requests.

If you intend to write AOPs, you can write them in either assembler or REXX. Assembler applications issue requests to the OM API; REXX applications issue REXX host commands to communicate with OM.

IMS provides a REXX SPOC API, which is a REXX program interface to a SPOC application. Your existing REXX applications can use this REXX SPOC API to interact with OM.

Related concepts

"CSL SCI requests" on page 200
SCI requests can be issued by an IMSplex member. Any member can also receive messages from any other IMSplex member after a connection is established.

"Sequence for coding CSL requests" on page 101
Most Common Service Layer (CSL) requests must be issued in a certain sequence.

Related reference

"CSL Operations Manager XML output" on page 233
Command responses that are returned through the OM API are embedded in XML tags using codepage 037. XML output is generated for responses to the CSLOMI, CSLOMCMD, and CSLOMQRY requests.

How AOP clients that run on the host communicate with the CSL OM

Automated operator program (AOP) clients that run on the host can communicate directly with Operations Manager (OM). After a z/OS AOP is registered with SCI, it can issue an OM command (CSLOMCMD) or

query (CSLOMQRY) requests. When the z/OS AOP is ready to terminate, it must deregister with SCI using the CSLSCDRG macro. Each of the requests can be sent directly to OM or SCI.

The following table lists the sequence of requests that are issued from an AOP that is running on the host. The request is listed with its purpose.

Table 35. Sequence of requests for an AOP OM client running on the host

Request	Purpose
CSLSCREG	Registers to SCI, which enables the client to send OM requests to OM through SCI.
CSLSCRDY	Readies the OM client to SCI, which routes messages to the client by client type.
CSLOMxxx	Issues OM requests (CSLOMCMD, CSLOMQRY) to send commands to OM.
CSLSCBFR	Releases the output buffer returned by the request, if any.
CSLSCQSC	Quiesces with SCI.
CSLSCDRG	Deregisters from SCI.

Note: Although not required for an AOP executing on the host, CSLSCRDY and CSLSCQSC are recommended for clients that want to receive messages routed by TYPE.

An OM client uses OM requests to access and use OM services and resources. Some SCI and OM requests must be issued by the client to request OM services. Some of those requests must be issued in a particular sequence, as shown in Table 35 on page 126. Other requests can be issued multiple times, in any order, based on the processing requirements of the client.

How AOP clients that run on a workstation communicate with the CSL OM

A workstation automated operator program (AOP) client cannot communicate directly with Operations Manager (OM). Instead, it must communicate with a z/OS address space that acts as an OM AOP client.

Instead of issuing CSLOMCMD or CSLOMQRY requests, the z/OS address space issues CSLOMI, which passes the prebuilt string that it received from the workstation to OM. For example, if the workstation wants to query the command processing clients to see how many exist in the IMSplex, it can send the string QUERY(CMDCLIENTS) to the z/OS address space, which would then use CSLOMI to send the query to OM for command processing.

If the workstation wants to issue a QRY TRAN command to the IMSplex, it can send the following string to the z/OS address space:

```
CMD(QUERY TRAN NAME) ROUTE(IMSA) TIMEOUT(10) RQSTTKN2(QTRANCMD)
```

The z/OS address space would then use CSLOMI to send the string to OM for command processing. The z/OS address space should pass the user ID associated with the workstation application to ensure correct authorization processing by OM.

The following table illustrates the sequence of requests issued by a proxy AOP client, executing on z/OS, that is communicating with OM on behalf of a workstation AOP. The request is listed with its purpose.

Table 36. Sequence of requests for AOP running on the workstation

Request	Purpose
CSLSCREG	Registers to SCI, which enables the client to send OM requests to OM through SCI.
CSLSCRDY	Readies the OM client to SCI, which routes messages to the client by client type.
CSLOMI	Issues OM requests (CMD(), QUERY()) to send commands to OM.
CSLSCBFR	Releases the output buffer returned by the request, if any.
CSLSCQSC	Quiesces with SCI.

Table 36. Sequence of requests for AOP running on the workstation (continued)

Request	Purpose
CSLSCDRG	Deregisters from SCI.

Note: Although not required for an AOP executing on the workstation, CSLSCRDY and CSLSCQSC are recommended for clients that want to receive messages routed by TYPE.

Processing AOP commands with a command processing client

A command processing client, such as an IMS control region, is a system that provides a command processor to accept and process commands entered by an automated operator program (AOP).

A command processing client must register to OM in addition to registering with SCI. The command processing client registers with OM by passing a list of commands to OM that it can process.

After successful command registration, the client must inform OM that it is ready to process commands.

Because AOP commands can be routed through any active OM in an IMSplex, a command processing client must register its command list and ready status with all active OMs. Registering with all OMs in the IMSplex ensures that any AOP command intended for the command processing client will be routed correctly, regardless of the OM that routes that command.

Like the AOP clients, command processing clients must issue requests in a particular sequence. This sequence, and the purpose of the request, is listed in Table 37 on page 127.

Table 37. Sequence of requests for a command processing client

Request	Purpose
CSLSCREG	Registers to SCI, which enables the client to send OM requests to OM through SCI.
CSLSCRDY	Readies the OM client to SCI, which routes messages to the client by client type.
CSLOMREG	Registers the command list to OM.
CSLOMRDY	Readies OM client to OM. Client is now ready to process commands.
CSLOMRSP	Sends the command response output back to OM after receiving and processing a command from OM.
CSLOMDRG	Deregisters from OM. The client no longer wants to process commands.
CSLSCQSC	Quiesces with SCI.
CSLSCDRG	Deregisters from SCI.

Interpreting CSL OM XML output

Command responses that are returned through the OM API are embedded in XML tags. XML output is generated for responses to the CSLOMI, CSLOMCMD, and CSLOMQRY requests. The DSECTS for the output of CSLOMCMD and CSLOMI when OPTION=AOPOUTPUT are described in the DISPLAY macro in the IMS.SDFSMAC data set.

For example, with the CSLOMI request, the QUERY parameter allows you to query all clients that are registered to OM. The clients are returned embedded in <cmdclients></cmdclients> tags.

The list of XML tags and the descriptions of each tag are provided in “XML tags returned as CSL OM responses” on page 238.

Chapter 5. Writing a CSL ODBM client

You can write your own ODBM clients that register with Open Database Manager (ODBM) and use the CSLDMI interface to pass DL/I calls to databases that are managed by IMS DB in DBCTL and DB/TM systems within an IMSplex.

IMS Connect, which provides TCP/IP connection management and routing services for the IMS Universal drivers, is an example of an ODBM client. IMS Connect and the IMS Universal drivers are delivered with IMS and you do not need to code any CSL ODBM requests to use them with ODBM.

Other ODBM clients can be application servers, such asWebSphere® Application Server for z/OS or Db2 for z/OS, that run application programs that access IMS databases through the IMS ODBM API.

To write an ODBM client, you can use the set of client requests provided by ODBM. These requests allow the ODBM client, a z/OS application program written in the assembler programming language, to access IMS databases in an IMSplex that are managed by IMS DB systems configured for the IMS DBCTL or DB/TM environments. An example of an ODBM client is IMS Connect.

ODBM clients submit CSLDMI requests to register with ODBM, interact with IMS databases, and manage syncpoint processing for local or global transactions.

The ODBM CSLDMI API uses the IMS ODBA interface to communicate with IMS DB and therefore supports only the DL/I calls that the ODBA interface supports. ODBM clients pass the DL/I calls to IMS DB by using the ODBMCI function of the CSLDMI API.

Sequence of ODBM client requests

Some requests to Open Database Manager (ODBM) from an ODBM client must be issued in a particular sequence, such as when enabling or disabling communication with ODBM.

An ODBM client issues SCI and ODBM requests to request ODBM services. Some of the requests must follow a particular sequence. Other requests can be issued multiple times, in any order, based on the processing requirements of the client.

Before an ODBM client can issue ODBM requests, it must register with SCI and all active ODBMs in the IMSplex.

The following table shows the basic sequence of requests that an ODBM client issues. The CSLSCREG, CSLSCRDY, and CSLDMREG requests must be issued in the order shown. The CSLSCBFR, CSLDMDRG, and CSLSCDRG requests must also be issued in the order shown. The sequence order of the CSLDMI requests can vary depending on the requirements of the ODBM client and syncpoint processing.

Table 38. Sequence of requests for an ODBM client

Request	Purpose
CSLSCREG	Registers to SCI, which enables the client to send ODBM requests to ODBM through SCI.
CSLSCRDY	Readies the ODBM client to SCI, which routes messages to the client by client type.
CSLDMREG	Registers client to ODBM to enable communication with ODBM.
CSLDMI FUNC=ODBMCI DLIFUNC	Allocates and deallocates PSBs. Passes DL/I calls to IMS DB through ODBM.
CSLDMI FUNC=COMMIT	For a local unit of work, commits the updates associated with a single APSB call.

Table 38. Sequence of requests for an ODBM client (continued)

Request	Purpose
CSLDMI FUNC=BACKOUT	For a local unit of work, backs out the updates associated with a single APSB call.
CSLDMI FUNC=READYSYNCPT	For a global unit of work, prepares one of multiple APSB calls for syncpoint processing.
CSLSCBFR	Releases the output buffer returned by the request, if any.
CSLDMDRG	Deregisters client from ODBM to end communications with ODBM.
CSLSCQSC	Quiesces the ODBM client to SCI. SCI will no longer route to the client by client type.
CSLSCDRG	Deregisters from SCI.

Related tasks

"Registering an ODBM client" on page 99
To register with ODBM, a client must first register with the CSL SCI and then with all active ODBMs in the IMSplex.

Related reference

"CSLDMREG: ODBM client registration request" on page 141
The CSLDMREG request registers an ODBM client with ODBM.

CSL ODBM client requests

ODBM clients submit requests to register with ODBM, interact with IMS databases, and commit or backout database updates.

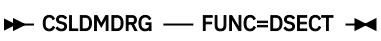
CSLDMDRG: ODBM client deregistration request

The CSLDMDRG request deregisters an ODBM client with ODBM.

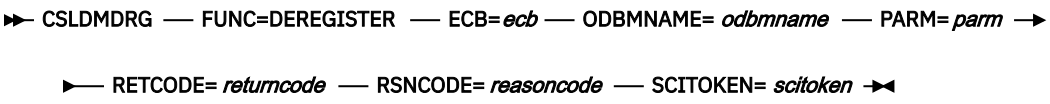
An ODBM client issues the CSLDMDRG request when the ODBM client is finished sending ODBA calls through ODBM. The deregister request cleans up the internal control blocks that ODBM stores for the ODBM client.

CSLDMDRG DSECT syntax

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



CSLDMDRG request protocol syntax



CSLDMDRG parameters

The CSLDMDRG parameters specify the ODBM values required for deregistration with ODBM.

The addresses can be specified as either a symbol or a register from 2 to 12.

The CSLDMDRG request includes the following parameters:

ODBMNAME=*symbol*
ODBMNAME=*(r2-r12)*
(Required) - Specifies the 8-byte ODBM name to which to send the command deregistration request.

If the value of ODBMNAME is specified as a symbol, the symbol must be the label of the ODBM field. If the value of ODBMNAME is specified as a register, the register must contain the address of the ODBM name field.

PARM=*symbol*
PARM=*(r1-r12)*
(Required) - Specifies the CSLDMDRG parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by `DDRG_PARMLN`.

If the value of PARM is specified as a symbol, the symbol must denote the start of the parameter list storage. If the value of PARM is specified as a register, the register must contain the address of the parameter list.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the return code on output. ODBM return codes are defined in the CSLDRR. SCI return codes are defined in CSLSRR.

The return code can be from ODBM (CSLDMDRG) or SCI (CSLSCMSG or CSLSCRQS). If ECB is specified, the RETCODE is not valid until the ECB is posted. All return codes contain the SCI member type indicator for either SCI or ODBM in the high order byte (X'01' for SCI and X'04' for ODBM).

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. ODBM reason codes are defined in the CSLDRR. SCI reason codes are defined in CSLSRR.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token is returned by a successful `CSLSCREG FUNC=REGISTER` request.

CSLDMDRG return and reason codes

The return and reason codes in the following table can be returned on a CSLDMDRG macro request.

Table 39. CSLDMDRG return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The CSLDMDRG request completed successfully.
X'04000010'	X'00004004'	The client is not registered.
	X'00004020'	The parameter list version of the parameter list specified on the PARM= parameter is invalid.

CSLDMI: ODBM application program interface

Application programs written in assembler and running on z/OS can access IMS databases managed by IMS DB in DBCTL and DB/DC systems in an IMSplex by using the CSL Open Database Manager (ODBM) CSLDMI API.

The CSLDMI API supports all of the DL/I calls supported by the IMS ODBA interface, global and local transaction processing, and security.

Prior to issuing the CSLDMI request, the ODBM client must first register with ODBM by issuing the CSLDMREG request.

The CSLDMI API includes the following function calls:

BACKOUT
Backs out a local unit of work for local transactions. A local unit of work consists of a single APSB call that has work, such as DL/I calls, associated with the APSB thread.

COMMIT
Commits a local unit of work for local transactions. A local unit of work consists of a single APSB call that has work, such as DL/I calls, associated with the APSB thread.

DSECT
Includes equate (EQU) statements in an application program for the length of a CSLDMI parameter list and for CSLDMI return and reason codes.

ODBMCI
To issue DL/I calls to IMS databases, use the ODBMCI function. DL/I calls are passed to IMS by using the DLIFUNC parameter of the ODBMCI function.

Prior to issuing an ODBMCI function call, you must ensure that the AIB fields are coded appropriately for the DL/I calls that are being passed in the DLIFUNC parameter.

The following parameters are specific to the ODBMCI function and cannot be specified on other CSLDMI function calls:

- AIB
- CLIENTID
- CLIENTIDLEN
- CTXTOKEN
- DLIFUNC
- GROUPNAME=
- GROUPNAMELEN=
- IOAREA
- IOAREALEN
- PCB
- PCBLEN
- SECTKNLEN
- SECTOKEN
- SSA1 through SSA15
- SSA1LEN through SSA15LEN
- URTOKEN
- USERID=
- USERIDLEN=

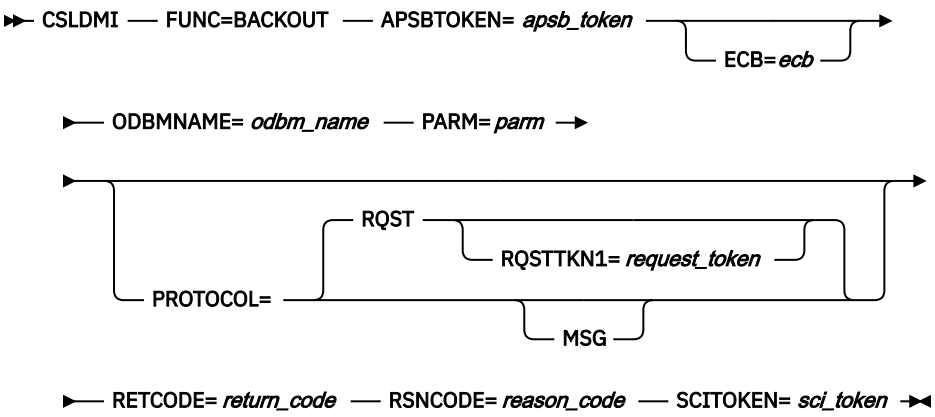
READYSYNCPT
Prepares for syncpoint processing for each of the multiple APSB calls within a global unit of work.

The READYSYNCPT function of the CSLDMI API must called before initiating syncpoint processing for global transactions, such as those that contain multiple APSB threads within a single unit of work. The ODBM client must issue FUNC=READYSYNCPT for each APSB that is represented by an APSBTOKEN in a global unit of work (UOW). This pertains to the URTOKEN parameter. See the URTOKEN parameter for more description.

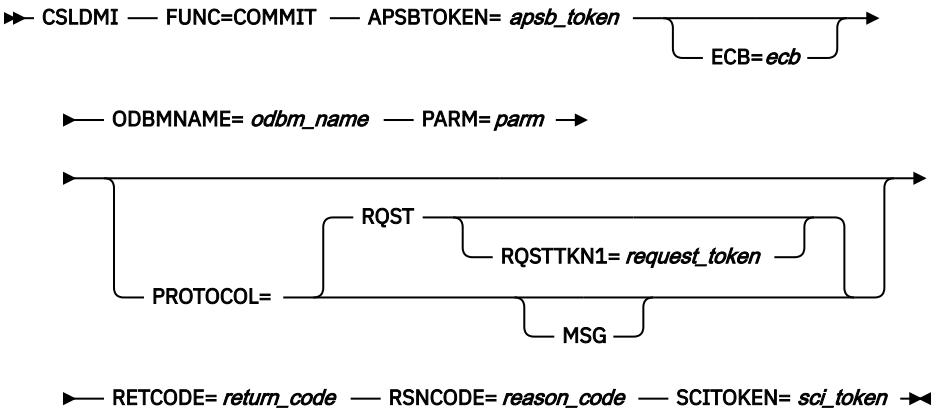
Before initiating syncpoint processing, the caller must issue FUNC=READYSYNCPT for each APSB call in the global UOW. CSLDMI uses an APSB token (APSBTOKEN) to represent each APSB call.

Subsections:

CSLDMI FUNC=BACKOUT syntax



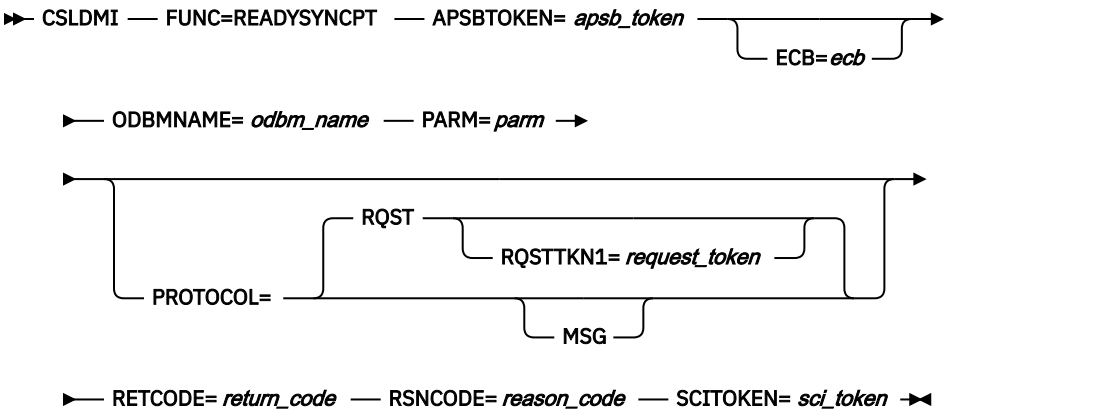
CSLDMI FUNC=COMMIT syntax



CSLDMI FUNC=DSECT syntax

CSLDMI FUNC=ODBMCI syntax

CSLDMI FUNC=READYSYNCPT syntax



CSLDMI function parameters

The CSLDMI parameters specify the ODBM values required for communicating with ODBM and accessing IMS databases.

The addresses can be specified as either a symbol or a register from 2 to 12.

Parameters that are supported only when the ODBMCI function is specified are noted in the description of the parameters. Parameters that are not noted as being supported by ODBMCI only, can be specified on all CSLDMI functions other than DSECT.

The following parameters can be specified on one or more functions of the CSLDMI API:

AIB=*symbol* | *(r2-r12)*
(Required) - Specifies the address of the application interface block (AIB). The AIB parameter is required on both input to and output from ODBM. This parameter is supported only on the ODBMCI function call.

When the SCI message protocol (PROTOCOL=MSG) is specified on a CSLDMI request, ODBM returns the address of the AIB in the DDIR_DMIRAIBPTR field of the ODBM directive parameter list, which is mapped by the CSLDMDIR macro.

Prior to issuing CSLDMI FUNC=ODBMCI, the AIB fields required by each DL/I call being passed on the DLIFUNC parameter must be set. The fields of the AIB are mapped by the DFSAIB macro and described in "Specifying the AIB mask for ODBA applications" in *IMS Version 15.3 Application Programming*.

If the AIB is specified as a register, the register must contain the address of the AIB area. If the AIB is specified as a symbol, the symbol must be the label of the AIB area.

APSBTOKEN=*symbol* | *(r2-r12)*
(Required) - Specifies an address for a 16-byte ODBM APSB token. An APSB token is returned by CSLDMI on the initial APSB thread request and is required on all subsequent calls targeted to this thread.

On an APSB request, APSBTOKEN specifies the address of the field to receive the token from ODBM. The length of APSBTOKEN field must be 16 bytes as defined by DMI_APSBTKNLEN in the CSLDMI macro.

For all subsequent requests associated to this thread, APSBTOKEN specifies the address of this 16-byte token.

When the SCI message protocol (PROTOCOL=MSG) is specified on a CSLDMI request, ODBM returns the address of the APSB token in the DDIR_DMIRAPSBTPTR field of the ODBM directive parameter list, which is mapped by the CSLDMDIR macro.

If the APSBTOKEN parameter is specified as a register, the register must contain the address of the token field. If the APSBTOKEN parameter is specified as a symbol, the symbol must be the label of the token field.

CLIENTID=*symbol* | *(r2-r12)*
(Optional) - Specifies the end user client application ID as defined by the end user client.

If the CLIENTID parameter is specified as a register, the register must contain the address of the client ID field. If the CLIENTID parameter is specified as a symbol, the symbol must be the label of the client ID field.

CLIENTIDLEN=*symbol* | *(r2-r12)*
(Required when CLIENTID is specified) - Specifies the length of the client ID.

If the CLIENTIDLEN parameter is specified as a register, the register must contain the length of the client ID. If the CLIENTIDLEN parameter is specified as a symbol, the symbol must be the label of a fullword containing the length of the client ID.

CTXTOKEN=*symbol* | *(r2-r12)*
(Optional) - Specifies a 16-byte RRMS Context Services context token. This may be the native context token, or a private token obtained by calling the Context Services Begin_Context service (CTXBEGC).

CTXTOKEN can be used to setup a global unit of work made up of multiple APSB calls within one commit scope on a single LPAR. If specified, the same CTXTOKEN must be included on each APSB call within the global unit of work.

CTXTOKEN and URTOKEN will be mutually exclusive if you do not install the PTF PH23803. After you install PH23803, if the call only specifies a CTXTOKEN, caller must ensure that the private context token specified on the CTXTOKEN has been disassociated from the current dispatchable unit, such as a TCB, prior to passing the private context token to ODBM. This is done using Context Services Switch_Context (CSXSWCH). ODBM associates this context to its DU for the duration of the call and disassociates it on return to the ODBM client.

If the caller specifies a CTXTOKEN and a URTOKEN, the context token can be native or private. The caller does NOT need to disassociate from the current DU. ODBM will express interest in the CTXTOKEN and create a cascaded UR child. The caller is responsible for initiating sync point processing using the appropriate z/OS Resource Recovery Services services, such as, ATRAPRP, ATRACMT, ATRCMIT, ATRBACK, and so forth. The **ODBM client** must establish an RRS WID (work identifier) if needed. Usually, the WID (Work Identifier) is an XID. ATRSWID2 is used to set a work identifier and the WID is set based on the context token specified on this CTXTOKEN parameter.

If the CTXTOKEN parameter is specified as a register, the register must contain the address of the private context token field. If the CTXTOKEN parameter is specified as a symbol, the symbol must be the label of the private context token field.

DLIFUNC=*symbol* | *(r2-r12)*

(Required) - Specifies the 4-byte DL/I call. Any DL/I call that is supported by the Open Database Access (ODBA) callable interface can be specified. This parameter is supported only on the ODBMCI function call.

Prior to issuing CSLDMI FUNC=ODBMCI DLIFUNC, the AIB fields required by each DL/I call being passed on the DLIFUNC parameter must be set. The fields of the AIB are mapped by the DFSAIB macro and described in "Specifying the AIB mask for ODBA applications" in *IMS Version 15.3 Application Programming*.

If the DLIFUNC parameter is specified as a register, the register must contain the address of the DLI function code. If the DLIFUNC parameter is specified as a symbol, the symbol must be the label of the DLI function code.

ECB=*symbol* | *(r2-r12)*

(Optional) - Specifies an MVS™ event control block (ECB) that is used for asynchronous requests. When the request is complete, the ECB specified is posted.

If an ECB is not specified, the task is suspended until the request is complete. If an ECB is specified, the ODBM client that invokes the CSLDMI macro must invoke the z/OS WAIT macro (or equivalent) after receiving control from CSLDMI before using or examining any data returned by CSLDMI, including the RETCODE and RSNCODE fields.

If the ECB parameter is specified as a register, the register must contain the address of the ECB. If the ECB parameter is specified as a symbol, the symbol must denote the start of the ECB storage.

GROUPNAME=*symbol* | *(r2-r12)*

(Optional) - Specifies a group name for RACF or an equivalent security product. The group name pertains to the APSB call only (DLIFUNC=APSB) and is ignored for all other DL/I calls.

If the GROUPNAME parameter is specified as a register, the register must contain the address of the group name field. If the GROUPNAME parameter is specified as a symbol, the symbol must be the label of the group name field.

GROUPNAMELEN=*symbol* | *(r2-r12)*

(Required when GROUPNAME is specified) - Specifies the length of the group name.

If the GROUPNAMELEN parameter is specified as a register, the register must contain the length of the group name. If the GROUPNAMELEN parameter is specified as a symbol, the symbol must be the label of a fullword containing the length of the group name.

IOAREA=*symbol* | *(r2-r12)*

(Conditionally required) - Specifies an I/O area that is used for the input or output data related to a database DL/I call. The IOAREA and IOAREALEN parameters are required only when a DL/I call that

requires input data or that returns output data is specified on the DLIFUNC parameter. This parameter is supported only on the ODBMCI function call.

When the SCI message protocol (PROTOCOL=MSG) is specified on a CSLDMI request, ODBM returns the address of the I/O area in the DDIR_DMIRIOAPT field of the ODBM directive parameter list, which is mapped by the CSLDMDIR macro.

If the IOAREA parameter is specified as a register, the register must contain the address of the I/O area. If the IOAREA parameter is specified as a symbol, the symbol must be the label of the I/O area.

IOAREALEN=*symbol* | *(r2-r12)*
(Conditionally required) - Specifies the length of the I/O area specified by the IOAREA parameter. The IOAREA and IOAREALEN parameters are required only when a DL/I call that requires input or that returns output is specified on the DLIFUNC parameter. This parameter is supported only on the ODBMCI function call.

The length specified on the IOAREALEN parameter must also be specified on the AIBOALEN field of the AIB mask. For information about the AIB mask, see Specifying the AIB mask (Application Programming) .

If the IOAREALEN parameter is specified as a register, the register must contain the length of the I/O area. If the IOAREALEN parameter is specified as a symbol, the symbol must be the label of a fullword containing the length of the I/O area.

ODBMNAME=*symbol* | *(r2-r12)*
(Required) - Specifies the 8-byte ODBM name to which to the CSLDMI request is to be sent.

If the ODBMNAME parameter is specified as a symbol, the symbol must be the label of the ODBM field. If the ODBMNAME parameter is specified as a register, the register must contain the address of the ODBM name field.

PARM=*symbol* | *(r2-r12)*
(Required) - Specifies the address CSLDMI parameter list. The length of the parameter list must be at least as long as the value assigned to DMI_PARMLN in the CSLDMI macro.

Use CSLDMI FUNC=DSECT to include equate (EQU) statements in your application program for the length of the CSLDMI parameter list

If the value of PARM is specified as a register, the register must contain the address of the parameter list. If the value of PARM is specified as a symbol, the symbol must denote the start of the parameter list storage.

PCB=*symbol* | *(r2-r12)*
(Optional) - An output parameter that specifies the address of a fullword storage area to receive the address of the program communication block (PCB) returned by IMS after processing a DL/I call. The PCB contains the status codes related to a DL/I call and other fields. This parameter is supported only on the ODBMCI function call.

The PCB storage is not preallocated by the caller. Upon return from the request, the address in the PCB parameter contains the address of a storage buffer that contains the PCB. After the ODBM client is finished with the PCB, the ODBM client must release the PCB storage buffer by issuing the SCI request CSLSCBFR FUNC=RELEASE.

When the SCI message protocol (PROTOCOL=MSG) is specified on a CSLDMI request, ODBM returns the address of the PCB storage in the DDIR_DMIRPCBPTR field of the ODBM directive parameter list, which is mapped by the CSLDMDIR macro.

If specified as a register, the register must contain the address of a fullword to contain the address of the PCB. If specified as a symbol, the symbol must be the label of a fullword to contain the address of the PCB.

PCBLEN=*symbol* | *(r2-r12)*
(Conditionally required) - Specifies the length of the PCB returned by DL/I call processing. The PCBLEN parameter is required when the PCB parameter is specified. This parameter is supported only on the ODBMCI function call.

If specified as a register, the register must contain the address of a fullword to contain the length of the PCB. If specified as a symbol, the symbol must be the label of a fullword to contain the length of the PCB.

PROTOCOL=MSG | RQST
(Optional) - Specifies the SCI protocol for sending the request to ODBM.

MSG
Specifies that CSLDMI sends input requests to ODBM by using the SCI message protocol, which uses a one-way send of the data to the other IMSplex members and does not support output parameters. The ODBM client does not wait for output from ODBM and any output generated by the requests that use the MSG protocol is handled asynchronously.

RQST
Specifies that CSLDMI sends input requests to ODBM by using the SCI request protocol. The SCI request protocol supports both input and output parameters. The ODBM client waits for output from ODBM and process it synchronously. RQST is the default.

RETCODE=*symbol* | *(r2-r12)*
(Required) - Specifies a 4-byte field to receive the return code on output. ODBM return codes are defined in the CSLDRR. SCI return codes are defined in CSLSRR.

The return code can be from ODBM (CSLDMI) or SCI (CSLSCMSG or CSLSCRQS). If an ECB is specified, the value of RETCODE is not valid until the ECB is posted. All return codes contain the SCI-member-type indicator for either SCI or ODBM in the high order byte (X'01' for SCI or X'04' for ODBM).

RQSTTKN1=*symbol* | *(r2-r12)*
(Conditionally optional) - Specifies a 16-byte user-generated request token that correlates an output response to its associated input request for asynchronous processing. ODBM returns the address of this token in the DDIR_DMIRQT1PTR field of the ODBM directive parameter list that is mapped by CSLDMDIR.

RQSTTKN1 is supported only when the SCI message protocol is specified by PROTOCOL=MSG.

If specified as a register, the register must contain the address of the request token field. If specified as a symbol, the symbol must be the label of the request token field. The request token field must be 16 bytes in length, left justified, and padded with blanks if necessary.

RSNCODE=*symbol* | *(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. ODBM reason codes are defined in the CSLDRR macro. SCI reason codes are defined in CSLSRR.

SCITOKEN=*symbol* | *(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token is returned by a successful `CSLSCREG FUNC=REGISTER` request.

If the SCITOKEN parameter is specified as a register, the register must contain the address of the SCI token field. If the SCITOKEN parameter is specified as a symbol, the symbol must be the label of the SCI token field.

SECTKNLEN=*symbol* | *(r2-r12)*
(Conditionally required) - Specifies the length of the security token. The SECTKNLEN parameter is required when the SECTOKEN parameter is specified. This parameter is supported only on the ODBMCI function call.

If the SECTKNLEN parameter is specified as a register, the register must contain the length of the security token. If the SECTKNLEN parameter is specified as a symbol, the symbol must be the label of a fullword containing the length of the security token.

SECTOKEN=*symbol* | *(r2-r12)*
(Conditionally optional) - Specifies the address of a variable length security token that is used for security checking by RACF or an equivalent security product. The security token applies only to the APSB DL/I call that is specified by DLIFUNC=APSB. The security token is ignored for all other DL/I calls. This parameter is supported only on the ODBMCI function call. This security token is used only if the client address space is an authorized caller. If the client address space is unauthorized, the user ID is obtained automatically from z/OS control blocks.

The security token must be a security object. For example, if RACF is used, the security token must be a RACO (RACF Object). ODBM invokes RACROUTE REQUEST=VERIFY,ENVIR=CREATE with ENVRIN= to establish a security environment with a RACF accessor environment element (ACEE) for the APSB thread. IMS uses the ACEE during ODBA or RAS security authorization for the PSB.

If the SECTOKEN parameter is specified as a register, the register must contain the address of the security token field. If the SECTOKEN parameter is specified as a symbol, the symbol must be the label of the security token field.

SSA*n*=*symbol* | *(r2-r12)*
(Optional) - Specifies the segment search arguments (SSAs) for a DL/I call. A maximum of 15 SSAs can be specified: SSA1 through SSA15. This parameter is supported only on the ODBMCI function call.

If the parameters SSA1 through SSA15 are specified as registers, each register must contain the address of an SSA. If the parameters SSA1 through SSA15 parameters are specified as symbols, each symbol must be the label of a fullword that contains the address of an SSA.

SSA*n*LEN=*symbol* | *(r2-r12)*
(Conditionally required) - Specifies the length of the corresponding SSA list area. For every SSA*n* parameter specified, a corresponding SSA*n*LEN parameter is required, specified as SSA1LEN up through SSA15LEN. This parameter is supported only on the ODBMCI function call.

If the parameters SSA1LEN through SSA15LEN are specified as registers, the registers must contain the length of the SSA list areas. If the parameters SSA1LEN through SSA15LEN are specified as symbols, the symbols must be the labels of a fullword containing the length of the SSA list areas.

URTOKEN=*symbol* | *(r2-r12)*
(Optional) - Specifies a 16-byte RRS parent unit of recovery (UR) token obtained by calling the RRS Express_UR_Interest service that supports cascaded transactions (ATREINT2 or higher).

A URTOKEN is required in order to setup a global unit of work made up of multiple APSB calls within one commit scope on a single LPAR. If specified, the same URTOKEN must be included on each initial APSB call within the global unit of work.

CTXTOKEN and URTOKEN will be mutually exclusive before you install the PTF PH23803. After you install PH23803, if the call only specifies a URTOKEN, caller must ensure that the UR token supports cascading. ODBM will create a new context and create a cascaded child UR. If the caller specifies a URTOKEN and a CTXTOKEN, ODBM will express interest in the CTXTOKEN, create a new context, and create a cascaded UR child. The caller is responsible for initiating sync point processing using the appropriate RRS services, such as, ATRAPRP, ATRACMT, ATRABCK, ATRCMIT and so forth. Caller must establish an RRS WID (work identifier) if needed. Usually, the WID is an XID. ATRSWID2 is used to set a work identifier. The WID is set based on the parent UR token specified on this URTOKEN parameter.

If the URTOKEN parameter is specified as a register, the register must contain the address of the RRS UR token field. If the URTOKEN parameter is specified as a symbol, the symbol must be the label of the RRS UR token field.

USERID=*symbol* | *(r2-r12)*
(Optional) - Specifies the user ID to be used by RACF or an equivalent security product. Use this parameter only if your client address space has been authorized for this request. If your client is not authorized, the user ID is obtained automatically from z/OS control blocks. The user ID pertains to the APSB call only (DLIFUNC=APSB) and is ignored for all other DL/I calls.

If the USERID parameter is specified as a register, the register must contain the address of the user ID field. If the USERID parameter is specified as a symbol, the symbol must be the label of the user ID field.

USERIDLEN=*symbol* | *(r2-r12)*
(Required when USERID is specified) - Specifies the length of the user ID.

If the USERIDLEN parameter is specified as a register, the register must contain the length of the user ID. If the USERIDLEN parameter is specified as a symbol, the symbol must be the label of a fullword containing the length of the user ID.

CSLDMI return and reason codes

The return and reason codes in the following table can be returned on a CSLDMI macro request.

Table 40. CSLDMI return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The CSLDMI request completed successfully.
X'04000004'	X'00001004'	The Input user exit rejected the request.
X'04000008'	X'00002018'	Invalid AIB parameter.
	X'0000201C'	Invalid value is specified in the AIBRSNM1 field of the AIB mask.
	X'00002020'	Unsupported DL/I function.
	X'00002024'	The Input user exit incorrectly set the AIBOALEN field of the IAB mask to a value that is greater than the value that is specified on the IOAREALEN parameter.
X'04000010'	X'00004000'	Unable to locate the alias name.
	X'00004004'	The client is not registered.
	X'00004008'	The data store was not acquired.

Table 40. CSLDMI return and reason codes (continued)

Return code	Reason code	Meaning
X'04000014'	X'00005004'	Unable to obtain an APSB control block.
	X'00005008'	Unable to obtain an AIB control block.
	X'00005014'	Hash table ADD failed for APSB block.
	X'00005018'	Hash table FIND failed for APSB block.
	X'00005034'	OUTPUT buffer allocation failed.
	X'00005040'	RRS0_ASSOCCTX failed.
	X'00005044'	RRS0_DISCTX failed.
	X'00005048'	RRS0_COMMIT3_DMIR failed.
	X'0000504C'	RRS0_SUSI failed.
	X'00005050'	RRS0_BACKOUT3_DMIR failed.
	X'00005054'	APSB token length error.
	X'00005058'	APSB hash table RELEASE failed.
	X'0000505C'	RRS0_ASSOCCTX3 failed.
	X'00005060'	RRS0_CASCADE3 failed.
	X'00005064'	RRS0_GETCTX2 failed.
	X'00005068'	RRS0_ENDCTX2 failed.
	X'0000506C'	SEC0_CREATE call failed.
	X'00005070'	SEC0_DELETE call failed.
	X'00005074'	An APSB call failed to schedule a PSB.
	X'00005078'	PSB Name error.
	X'00005080'	PAPL block allocation failed.
	X'00005084'	RRS not active for global transaction.
	X'0000507C'	Failed to obtain DMI IOA storage.
	X'00005094'	BPETCBSW to an ORRS TCB failed.
	X'00005098'	BPETCBSW to an ODRA TCB failed.
	X'0000509C'	Failed to obtain an ORRS TTE.

Related reference

Specifying the AIB mask for ODBA applications (Application Programming)

Database management (Application Programming APIs)

Database management call summary (Application Programming APIs)

CSLDMREG: ODBM client registration request

The CSLDMREG request registers an ODBM client with ODBM.

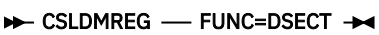
With the CSLDMREG request, an ODBM client, such asWebSphere Application Server for z/OS or Db2 for z/OS, can register with an instance of ODBM.

The CSLDMREG request registers an application program with ODBM as an ODBM client. The register command must be the first request that a client issues to ODBM.

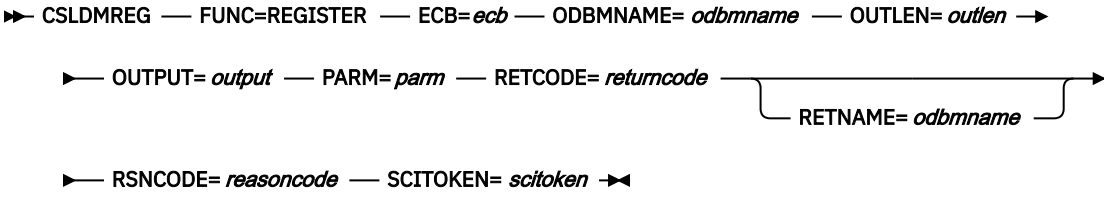
Use the CSLSCQRY request to obtain the names of all ODBMs in the IMSplex.

CSLDMREG DSECT syntax

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



Request protocol syntax



CSLDMREG parameters

The CSLDMREG parameters specify the ODBM values required for registration with ODBM.

The addresses can be specified as either a symbol or a register from 2 to 12.

The CSLDMREG command includes the following parameters:

ECB=*symbol*
ECB=*(r2-r12)*
(Optional) - Specifies the address of a z/OS event control block (ECB) used for asynchronous requests. When the request is complete, the ECB specified is posted. If an ECB is not specified, the task is suspended until the request is complete. If an ECB is specified, the invoker of the macro must issue a WAIT (or equivalent) after receiving control from CSLDMREG before using or examining any data returned by this macro (including the RETCODE and RSNCODE fields).

If the value of ECB is specified as a symbol, the symbol must denote the start of the ECB storage. If the value of ECB is specified as a register, the register must contain the address of the ECB.

ODBMNAME=*symbol*
ODBMNAME=*(r2-r12)*
(Optional) - Specifies the 8-byte ODBM name to which to send the command registration request.

Either the ODBMNAME parameter or the RETNAME parameter, but not both, must be specified on a CSLDMREG request.

Use the ODBMNAME parameter to connect the ODBM client to a specific, known instance of ODBM.

If the value of ODBMNAME is specified as a symbol, the symbol must be the label of the ODBM field. If the value of ODBMNAME is specified as a register, the register must contain the address of the ODBM name field.

OUTLEN=*symbol*
OUTLEN=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the length of the output returned by the CSLDMREG request. OUTLEN contains the length of the output pointed to by the OUTPUT= parameter.

If no output is built, the output length is zero, as is the case when an error is detected before building the output.

If the value of OUTLEN is specified as a register, the register must contain the address of the output length field. If the value of OUTLEN is specified as a symbol, the symbol must be the label of the output length field.

OUTPUT=*symbol*
OUTPUT=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the variable length output returned by the CSLDMREG request. The output length is returned in the OUTLEN= field.

If no output is built, the output address is zero, as is the case when an error is detected before building the output.

The CSLDREGO macro maps the output that is returned. The output buffer contains the ODBM version, count of the aliases, and a list of the 4-byte alias names.

The output buffer is not preallocated by the caller. Upon return from the request, this word contains the address of a buffer containing the update output.

After the ODBM client no longer needs the buffer storage, the ODBM client must release the storage by issuing the CSLSCBFR FUNC=RELEASE request.

If the value of OUTPUT is specified as a register, the register must contain the address of a field to contain the output address. If the value of OUTPUT is specified as a symbol, the symbol must be the label of a field to contain the output.

PARM=*symbol*
PARM=*(r1-r12)*
(Required) - Specifies the address of the CSLDMREG parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by DREG_PARMLN.

If the value of PARM is specified as a symbol, the symbol must denote the start of the parameter list storage. If the value of PARM is specified as a register, the register must contain the address of the parameter list.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the return code on output. ODBM return codes are defined in the CSLDRR. SCI return codes are defined in CSLSRR.

The return code can be from ODBM (CSLDMREG) or SCI (CSLSCMSG or CSLSCRQS). If ECB is specified, the RETCODE is not valid until the ECB is posted. All return codes contain the SCI member type indicator for either SCI or ODBM in the high order byte (X'01' for SCI and X'04' for ODBM).

If the value of RETCODE is specified as a symbol, the symbol must be the label of the return code field. If the value of RETCODE is specified as a register, the register must contain the address of the return code field.

RETNAME=*symbol*
RETNAME=*(r2-r12)*
(Optional) - Specifies the eight-byte output field to receive the ODBM name. This is the CSL member name of the target address space to which SCI sent the request.

Either the RETNAME parameter or the ODBMNAME parameter, but not both, must be specified on a CSLDMREG request.

Use the RETNAME parameter if you do not know the name of an ODBM instance or if the ODBM client does not require a connection to a specific instance of ODBM. When RETNAME is specified, SCI connects to any available instance of ODBM and returns the name of that ODBM in the RETNAME output field.

If the value of OUTPUT is specified as a register, the register must contain the address of a field to contain the output address. If the value of OUTPUT is specified as a symbol, the symbol must be the label of a field to contain the output.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. ODBM reason codes are defined in the CSLDRR. SCI reason codes are defined in CSLSRR.

If the value of RSNCODE is specified as a symbol, the symbol must be the label of the reason code field. If the value of RSNCODE is specified as a register, the register must contain the address of the reason code field.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token is returned by a successful `CSLSCREG FUNC=REGISTER` request.

If the value of SCITOKEN is specified as a register, the register must contain the address of the SCI token field. If the value of SCITOKEN is specified as a symbol, the symbol must be the label of the SCI token field.

CLSDMREG return and reason codes

The return and reason codes in the following table can be returned on a CSLDMREG macro request. The hexadecimal value 04 represents the SCI member type for ODBM.

Table 41. CSLDMREG return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The CSLDMREG request completed successfully.
X'04000010'	X'0000401C'	The client is already registered.
	X'00004020'	The parameter list version of the parameter list specified on the PARM= parameter is invalid.
X'04000014'	X'0000502C'	Unable to obtain a CSLDREGO feedback area.
	X'0000503C'	Unable to obtain a CLNT control block.

Related concepts

“Sequence of ODBM client requests” on page 129
Some requests to Open Database Manager (ODBM) from an ODBM client must be issued in a particular sequence, such as when enabling or disabling communication with ODBM.

Related tasks

“Registering an ODBM client” on page 99
To register with ODBM, a client must first register with the CSL SCI and then with all active ODBMs in the IMSplex.

Chapter 6. Writing a CSL OM client

These topics describe the client requests and directives for writing an OM client.

CSL OM command processing client requests

The following topics describe the requests that are made by command processing clients.

If you are writing your own command processing client, ensure that the access authority you provide on the RACF PERMIT command matches the access authority with which the command is registered.

Related tasks

"Registering an OM command processing client" on page 100
An Operations Manager (OM) command processing client must register its command with OM, whereas automated operator program (AOP) clients do not have to register to OM.

CSLOMBLD: command registration build

With the CSLOMBLD request, you can build the command registration list that is passed to OM on the CSLOMREG request.

This list identifies the commands for which the IMS system can be called. The list contains a set of statements starting with a CSLOMBLD FUNC=BEGIN statement and ending with a CSLOMBLD FUNC=END statement.

Any number of CSLOMBLD FUNC=DEFVRB statements can be provided, each one defining the command verb. Following each DEFVRB statement are CSLOMBLD FUNC=DEFKEY statements, which identify keywords valid for the previously defined command verb.

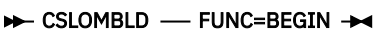
The set of CSLOMBLD statements can be defined either in a separate data-only assembler module, or in a static data section of an executable assembler module. Refer to the documentation in the CSLOMBLD macro.

CSLOMBLD is used to build the command registration list; it does not have an input parameter list.

CSLOMBLD syntax

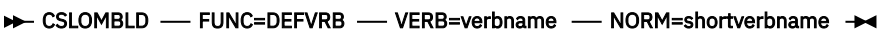
CSLOMBLD BEGIN

Use the BEGIN function statement to identify the beginning of the set of command statements.



CSLOMBLD DEFVRB

Use the DEFVRB function statement to identify a command that the OM client or IMS system will support. You can specify a short form of the command verb.



CSLOMBLD DEFKEY

Use the DEFKEY function statement to identify a keyword that is valid for the previously defined command. You can also specify command routing and required RACF authorization with this statement.

CSLOMBLD DEFGMR

Use the DEFGMR function statement to identify the beginning of the statements that describe the output parsing grammar.

Note: This function is for internal IBM use only.

```
▶▶ CSLOMBLD ── FUNC=DEFGMR ──▶◀
```

CSLOMBLD ENDGMR

Use the ENDGMR function statement to designate the end of the statements that describe the output parsing grammar.

Note: This function is for internal IBM use only.

```
▶▶ CSLOMBLD ── FUNC=ENDGMR ──▶◀
```

CSLOMBLD END

Use the END function statement to designate the end of the list of command statements.

```
▶▶ CSLOMBLD ── FUNC=END ──▶◀
```

CSLOMBLD parameters

KEYW=*keyword*
Specifies a valid keyword for the command verb that immediately precedes this parameter. For a null keyword, use blanks; for example, 'KEYW= '. This parameter is required for FUNC=DEFKEY.

NORM=*shortverbname*
Specifies the short form of the command being defined. This parameter is required for FUNC=DEFVRB.

ROUTE=*ANY | ALL*
Specifies the override routing for the command being defined. This parameter is required for FUNC=DEFKEY.

SEC=*READ | UPDATE*
Specifies the required RACF authorization for KEYW. This parameter is required for FUNC=DEFKEY.

VERB=*verbname*
Specifies the long form of the command being defined. This parameter is required for FUNC=DEFVRB.

CSLOMBLD example

The following shows an example of a set of CSLOMBLD statements.

Figure 7. CSLOMBLD example statements

```
CSLOMBLD FUNC=BEGIN
CSLOMBLD FUNC=DEFVRB,VERB=ACTIVATE,NORM=ACT
CSLOMBLD FUNC=DEFKEY,KEYW=LINK,SEC=UPDATE
CSLOMBLD FUNC=DEFKEY,KEYW=NODE,SEC=UPDATE
CSLOMBLD FUNC=END
```

Overriding CSL OM command routing with the ROUTE parameter

CSLOMBLD allows the command processing client to override the routing that you specify when you enter a command. There are a few commands that specify command routing overrides. OM overrides command routing when two command processing clients specify different routing overrides for the same command if:

- At least one command processing client specifies an override of ROUTE=ALL, then OM routes the command to all registered command processing clients.

- At least one command processing client specifies an override of ROUTE=ANY, and no command processing client has specified ROUTE=ALL, then OM routes the command to one of the registered command processing clients.
- No command processing clients have specified an override of ROUTE=ALL or ROUTE=ANY, then OM routes the command as specified by the user that entered the command.

When an OM command has a ROUTE parameter, IMS chooses the highest level IMS in the route list as the command master. For example, in an IMSplex configuration that includes an IMS 15.3 system with an IMS 15.3 CQS, and another IMS system with a previous version SCI, if an INIT OLC command (which is a ROUTE=ANY command) is issued, the IMS 15.3 system is selected as the command master.

For a list of the commands that can be issued with the ROUTE parameter, see Commands and keywords supported by the OM API (Commands).

Related reference

/SECURE command (Commands)

"CSLOMREG: command registration request" on page 151

With the CSLOMREG request, a command processing client like the IMS control region can register commands with an OM. The registration tells OM which commands that a client can process.

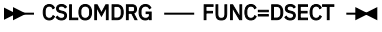
CSLOMDRG: command deregistration request

With the CSLOMDRG request, a command processing client like the IMS control region tells OM that it no longer wants to process commands. All client information from the OM command registration table is removed and OM stops sending further commands to the client.

CSLOMDRG syntax

DSECT syntax

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



Request protocol syntax

CSLOMDRG parameters

PARM=*symbol*
PARM=*(r2-r12)*

(Required) - Specifies the CSLOMDRG parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by ODRG_PARMLN.

RETCODE=*symbol*
RETCODE=*(r2-r12)*

(Required) - Specifies a 4-byte field to receive the return code on output. Possible return codes are described in the following table.

The return code can be from OM (CSLOMDRG) or SCI (CSLSCMSG or CSLSCRQS). If ECB is specified, the RETCODE is not valid until the ECB is posted. The value of the high-order byte in the return code identifies whether SCI (X'01') or OM (X'02') provided the return code. OM return codes are defined in the CSLORR. SCI return codes are defined in CSLSRR.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. OM reason codes are defined in the CSLORR. SCI reason codes are defined in CSLSRR. Possible reason codes are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token is returned by a successful CSLSCREG FUNC=REGISTER request.

CSLOMDRG return and reason codes

The return and reason codes in following table can be returned on a CSLOMDRG macro request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 42. CLSOMDRG return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.

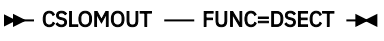
CSLOMOUT: unsolicited output request

The CSLOMOUT request is issued by a command processing client that wants to send a message indirectly in response to a command. The message can be additional information as a result of a command issued after the initial command response is returned to OM, or an informational message as a result of an event in the system. OM sends the unsolicited message to the OM Output user exit.

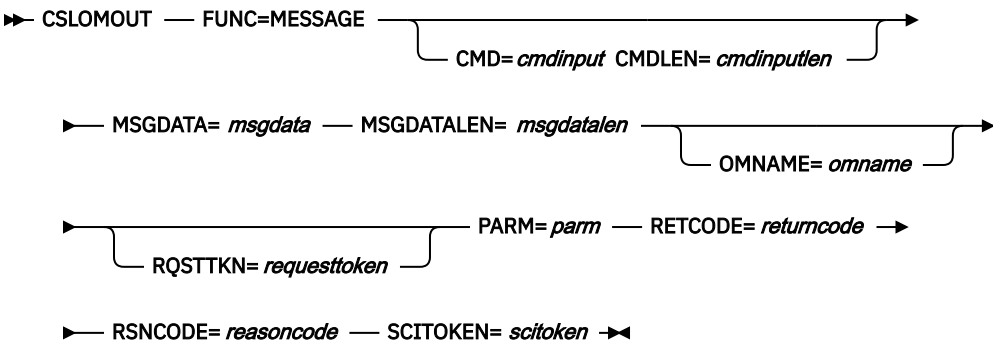
CSLOMOUT syntax

DSECT syntax

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



Request protocol syntax



CSLOMOUT parameters

CMD=*symbol*
CMD=*(r2-r12)*
(Optional) - Specifies the command input buffer. This can be any IMS command that can be specified through the OM API. This parameter represents the original command input.

CMDLEN=*symbol*
CMDLEN=*(r2-r12)*
(Optional) - Specifies the length of the command input buffer.

MSGDATA=*symbol*
MSGDATA=*(r2-r12)*
(Required) - Specifies the command response message buffer.

MSGDATALEN=*symbol*
MSGDATALEN=*(r2-r12)*
(Required) - Specifies the length of the command response message buffer.

OMNAME=*symbol*
OMNAME=*(r2-r12)*
(Optional) - Specifies the 8-byte OM name to which to send the unsolicited output message when the message is an asynchronous response to a command.

RQSTTKN=*symbol*
RQSTTKN=*(r2-r12)*
(Optional) - Specifies the 32-byte request token that was passed to the command processing client on an OM command directive. This represents the RQSTTKN1 and RQSTTKN2 fields that are entered on either or both the CSLOMI and CSLOMCMD requests.

PARM=*symbol*
PARM=*(r2-r12)*
(Required) - Specifies the CSLOMOUT parameter list. The length of the parameter list must be equal to the parameter list length EQU value that is defined by OOUT_PARMLN.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the return code on output. Possible return codes are described in the following table.

The return code can be from OM (CSLOMOUT) or SCI (CSLSCMSG or CSLSCRQS). If ECB is specified, the RETCODE is not valid until the ECB is posted. The value of the high-order byte in the return code identifies whether SCI (X'01') or OM (X'02') provided the return code. OM return codes are defined in the CSLORR. SCI return codes are defined in CSLSRR.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. OM reason codes are defined in the CSLORR. SCI reason codes are defined in CSLSRR. Possible reason codes are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token is returned by a successful CSLSCREG FUNC=REGISTER request.

CSLOMOUT return and reason codes

The return and reason codes in the following table can be returned on a CSLOMOUT macro request. The following table lists the return and reason codes that can be returned on a CSLOMI macro request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 43. CLSOMOUT return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.

CSLOMRDY: ready request

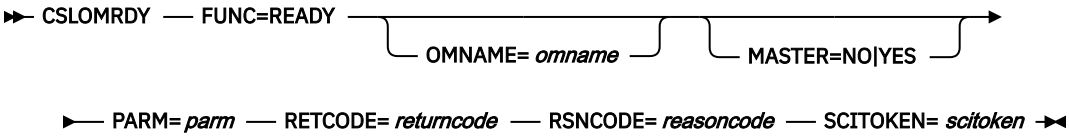
With the CSLOMRDY request, command processing clients like the IMS control region notify OM that they are ready to process commands. OM does not send commands to a client until this request is processed.

CSLOMRDY syntax

CSLOMRDY DSECT syntax

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

CSLOMRDY request protocol syntax



CSLOMRDY parameters

MASTER=NO
MASTER=YES
(Optional) - Specifies whether or not the client should be chosen as the command master. If a client specifies MASTER=YES, OM can select that client to be the command master. If the client specifies MASTER=NO, OM selects it to be the command master only if no other client has specified MASTER=YES.

OMNAME=*symbol*
OMNAME=*(r2-r12)*
(Optional) - Specifies the 8-byte OM name to which to send the command ready request. If an OM name is not specified, the ready request is sent to all OM address spaces registered in the IMSplex.

PARM=*symbol*
PARM=*(r2-r12)*
(Required) - Specifies the CSLOMRDY parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by ORDY_PARMLN.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the return code on output. Possible return codes are described in the following table.

The return code can be from OM (CSLOMRDY) or SCI (CSLSCMSG or CSLSCRQS). If ECB is specified, the RETCODE is not valid until the ECB is posted. The value of the high-order byte in the return code identifies whether SCI (X'01') or OM (X'02') provided the return code. OM return codes are defined in the CSLORR. SCI return codes are defined in CSLSRR.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. OM reason codes are defined in the CSLORR. SCI reason codes are defined in CSLSRR. Possible reason codes are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token is returned by a successful CSLSCREG FUNC=REGISTER request.

CSLOMRDY return and reason codes

The following table lists the return and reason codes that can be returned on a CSLOMRDY macro request. Also included is the meaning of the reason code (that is, what possibly caused it).

Table 44. CLSOMRDY return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.

Related concepts

CSL OM command routing (System Administration)

CSLOMREG: command registration request

With the CSLOMREG request, a command processing client like the IMS control region can register commands with an OM. The registration tells OM which commands that a client can process.

CSLOMREG must be the first request that a command processing client issues to OM. A command processing client must register with all OM address spaces in the IMSplex. If a client is registered with only one OM in an IMSplex, and that OM goes down, the client's commands are not routed to another OM in the IMSplex. Use the CSLSCQRY request to obtain the names of all OMs in the IMSplex. The client must be authorized to issue a CSLOMREG request. This authorization is from SCI, which notifies OM that the client can issue requests.

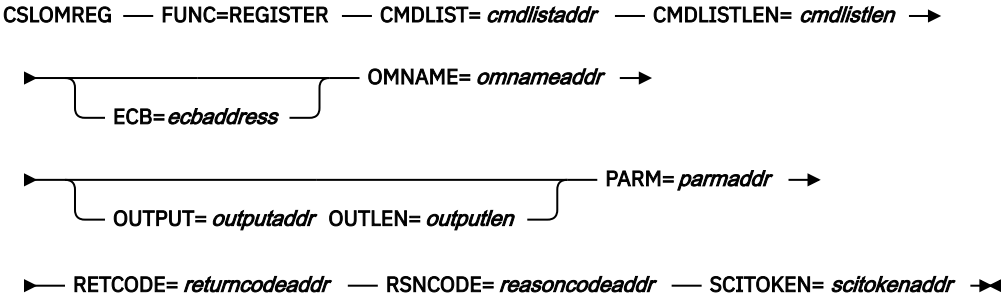
CSLOMREG syntax

CSLOMREG DSECT syntax

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

CSLOMREG request protocol syntax

The syntax for the CSLOMREG request follows.



CSLOMREG parameters

CMDLIST=*symbol*
CMDLIST=*(r2-r12)*
(Required) - Specifies the command definition list.

The command list is built using the CSLOMBLD macro.

CMDLISTLEN=*symbol*
CMDLISTLEN=*(r2-r12)*
(Required) - Specifies the length of the command definition list buffer.

ECB=*symbol*
ECB=*(r2-r12)*
(Optional) - Specifies the address of a z/OS event control block (ECB) used for asynchronous requests. When the request is complete, the ECB specified is posted. If an ECB is not specified, the task is suspended until the request is complete. If an ECB is specified, the invoker of the macro must issue a WAIT (or equivalent) after receiving control from CSLOMREG before using or examining any data returned by this macro (including the RETCODE and RSNCODE fields).

OMNAME=*symbol*
OMNAME=*(r2-r12)*
(Required) - Specifies the 8-byte OM name to which to send the command registration request.

OUTLEN=*symbol*
OUTLEN=*(r2-r12)*
(Optional) - Specifies a 4-byte field to receive the length of the output returned by the CSLOMREG request. OUTLEN contains the length of the output pointed to by the OUTPUT= parameter.

The output length is zero if no output is built, for example, if an error is detected before any output can be built.

OUTPUT=*outputaddr*
OUTPUT=*(r2-r12)*
(Required) - Specifies a field to receive the variable length output returned by the CSLOMREG request. The output length is returned in the OUTLEN= field.

The output is mapped by the CSLOREGO macro and is built only if there was an error registering one or more commands. The output contains a header and one or more list entries. Refer to the CSLOREGO macro for the output fields.

The output address is zero if no output was built, for example, if an error was detected before any output could be built.

The output buffer is not preallocated by the caller. After the request returns it, this word contains the address of a buffer containing the update output. It is the caller's responsibility to release this storage by issuing the CSLSCBFR FUNC=RELEASE request when it is finished with the storage. The length of the output is returned in the OUTLEN=field.

PARM=*symbol*
PARM=*(r2-r12)*
(Required) - Four-byte input parameter that specifies the address of the storage used by the request to pass the parameters to SCI. The length of the parameter list must be equal to the parameter list length EQU value defined by OREG_PARMLN.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the return code on output.Possible return codes are described in the following table.

The return code can be from OM (CSLOMREG) or SCI (CSLSCMSG or CSLSCRQS). If ECB is specified, the RETCODE is not valid until the ECB is posted. The value of the high-order byte in the return code identifies whether SCI (X'01') or OM (X'02') provided the return code. OM return codes are defined in the CSLORR. SCI return codes are defined in CSLSRR.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. OM reason codes are defined in the CSLORR. SCI reason codes are defined in CSLSRR. Possible reason codes are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token is returned by a successful CSLSCREG FUNC=REGISTER request.

You can find the return and reason codes for the CSLOMREG command request in CSLOMREG return and reason codes (Messages and Codes).

Related reference

"CSLOMBLD: command registration build" on page 145
With the CSLOMBLD request, you can build the command registration list that is passed to OM on the CSLOMREG request.

CSLOMRSP: command response request

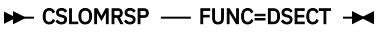
The CSLOMRSP request is issued by a command processing client in response to a command. Command response information is consolidated and sent to OM.

A command processing client issues the CSLOMRSP request in response to a command. All command response information from an individual command processing client is consolidated by the client and sent to OM in one request. OM consolidates the responses from multiple clients into one response for the automated operator program client.

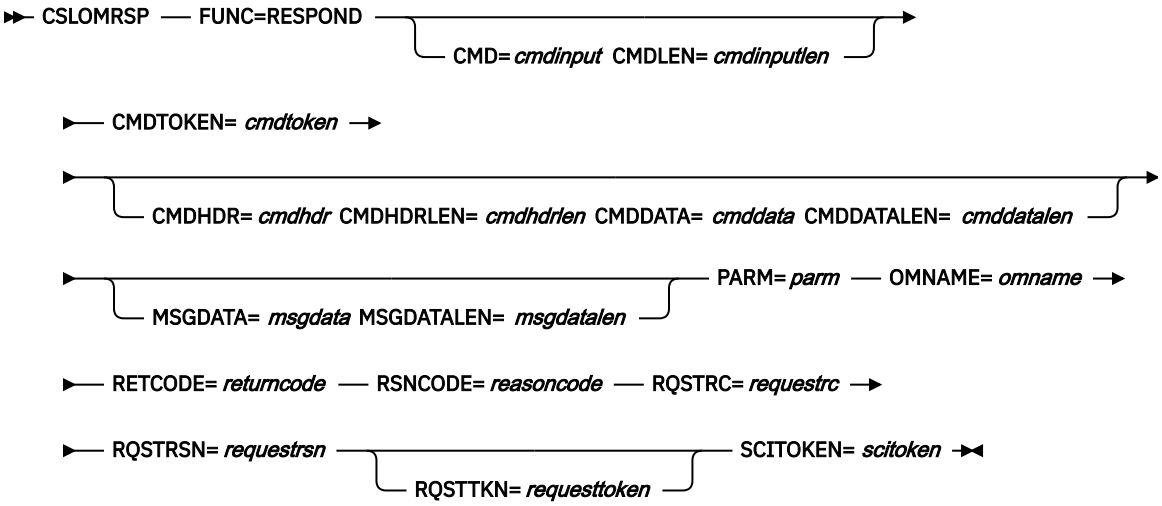
CSLOMRSP syntax

CSLOMRSP DSECT syntax

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



CSLOMRSP request protocol syntax



CSLOMRSP parameters

CMD=*symbol*
CMD=*(r2-r12)*

(Optional) - Specifies the command input buffer. This can be any IMS command that can be specified through the OM API.

This parameter is optional; what you provide here will be included in the input tags that are returned as XML output.

CMDDATA=*symbol*
CMDDATA=*(r2-r12)*

(Optional) - Specifies the command response data buffer.

CMDDATALEN=*symbol*
CMDDATALEN=*(r2-r12)*
(Optional) - Specifies the length of the command response data buffer.

CMDHDR=*symbol*
CMDHDR=*(r2-r12)*
(Optional) - Specifies the command response header buffer.

CMDHDRLEN=*symbol*
CMDHDRLEN=*(r2-r12)*
(Optional) - Specifies the length of the command response header buffer.

CMDLEN=*symbol*
CMDLEN=*(r2-r12)*
(Optional) - Specifies the length of the command input buffer.

CMDTOKEN=*symbol*
CMDTOKEN=*(r2-r12)*
(Required) - Specifies a 32-byte field to contain the command token. This token uniquely identifies the command instance that the client has processed. The command token is passed to the client on an OM command directive. The address of the token is passed to the client in the ODIR_CMDTKPTR field in the OM command directive parameter list.

MSGDATA=*symbol*
MSGDATA=*(r2-r12)*
(Optional) - Specifies the command response message buffer.

MSGDATALEN=*symbol*
MSGDATALEN=*(r2-r12)*
(Optional) - Specifies the length of the command response message buffer.

PARM=*symbol*
PARM=*(r2-r12)*
(Required) - Four-byte input parameter that specifies the address of the storage used by the request to pass the parameters to SCI. The length of the parameter list must be equal to the parameter list length EQU value defined by ORSP_PARMLN.

OMNAME=*symbol*
OMNAME=*(r2-r12)*
(Required) - Specifies the 8-byte OM name to which to send the command registration request.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the return code on output. Possible return codes are described in the following table.

The return code can be from OM (CSLOMRSP) or SCI (CSLSCMSG or CSLSCRQS). If ECB is specified, the RETCODE is not valid until the ECB is posted. The value of the high-order byte in the return code identifies whether SCI (X'01') or OM (X'02') provided the return code. OM return codes are defined in the CSLORR. SCI return codes are defined in CSLSRR.

RQSTRC=*symbol*
RQSTRC=*(r2-r12)*
(Required) - Specifies a 4-byte field to contain the reason code to be passed to the originator of the command. This reason code is defined by the command processing client and indicates the result of the command. Non-zero reason codes are passed back to the client in the <cmderr> section of the command response.

RQSTRSN=*symbol*
RQSTRSN=*(r2-r12)*
(Required) - Specifies a 4-byte field to contain the reason code to be passed to the originator of the command. This reason code is defined by the command processor client and indicates the result of the command. Non-zero reason codes are passed back to the client in the <cmderr> section of the command response.

RQSTTKN=*symbol*
RQSTTNK=*(r2-r12)*
(Optional) - Specifies the 32-byte request token that was passed to the command processing client on an OM command directive. This parameter represents the RQSTTKN1 and RQSTTKN2 fields that are entered on either or both the CSLOMI and CSLOMCMD requests.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. OM reason codes are defined in the CSLORR. SCI reason codes are defined in CSLSRR. Possible reason codes are described in the following table.

RSNTEXT=*symbol*
RSNTEXT=*(r2-r12)*
(Optional) - Specifies a register that must contain the address of the reason text buffer; if specified as a symbol it must be the label of the reason text buffer. The buffer consists of a two-byte length field followed by the reason text. This token allows an OM client to pass a text description of the reason code in a command response.

RSNTEXTLEN=*symbol*
RSNTEXTLEN=*(r2-r12)*
(Optional) - Specifies a register that must contain the address of the reason text buffer; if specified as a symbol it must be the label of the reason text buffer. The buffer consists of a two-byte length field followed by the reason text. This token allows an OM client to pass a text description of the reason code in a command response.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token is returned by a successful CSLSCREG FUNC=REGISTER request.

CSLOMRSP return and reason codes

The following table lists the return and reason codes that can be returned on a CSLOMRSP macro request. Also included is the meaning of the reason code (that is, what possibly caused it).

Table 45. CLSOMRSP return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.

Related reference

"CSL Operations Manager XML output" on page 233
Command responses that are returned through the OM API are embedded in XML tags using codepage 037. XML output is generated for responses to the CSLOMI, CSLOMCMD, and CSLOMQRY requests.

CSLOMSUB: Subscribe to unsolicited messages

With the CSLOMSUB request, single point of control (SPOC) clients can subscribe to OM in order to receive unsolicited output messages from a command processing client, such as IMS.

A message is defined as unsolicited when it is not generated as a response to an input message. For example, system informational messages are unsolicited messages. In an IMS 15.3 IMSplex environment, a command response is considered an unsolicited message if it is returned to a SPOC from which the command did not originate.

All unsolicited output messages are routed to the OM's Output exit routine, which can modify the output. SPOCs can subscribe to the OM using the CSLOMSUB request to have the unsolicited output messages, modified or not, routed to them. To identify unsolicited output messages that should not be directed to OM, you can edit a table that includes messages from OM, RM, SCI, CQS, and the IMS control region that should be ignored.

You must specify an SCI Input exit routine on the SCI registration request (CSLSCREG) to receive unsolicited output messages.

To understand if OM has terminated, specify an SCI Notify exit routine on the SCI registration request, specifying either the RETNAME or RETTOKEN parameter. The SCI Notify exit is called whenever there is a change in status of any member of the IMSplex. Save the RETNAME or RETTOKEN value from the SCI registration and use it to look for a match that indicates that the OM has terminated. The SPOC client can then subscribe to another OM.

This request is supported in Assembler and PLX.

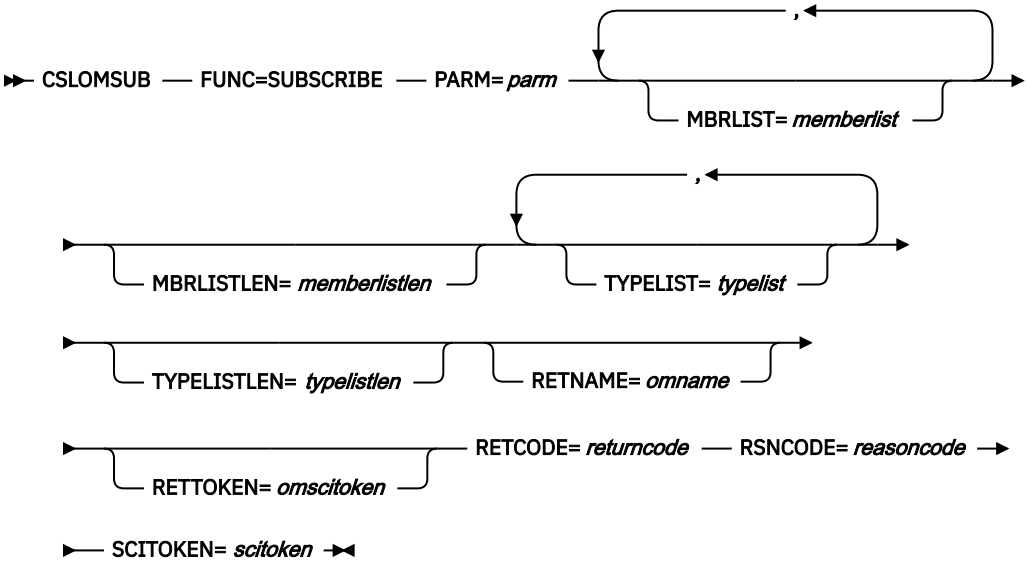
CSLOMSUB syntax

CSLOMSUB FUNC-DSECT syntax

Use the DSECT function of the CSLOMSUB request to include equate (EQU) statements in your logic for the CSLOMSUB parameter list length and return and reason codes.

CSLOMSUB request syntax

With this syntax, SPOC clients can subscribe to OM to receive unsolicited output messages.



CSLOMSUB request parameters

PARM=*parm*
PARM=*(r1-r12)*
Specifies the CSLOMSUB parameter list. The length of the parameter list must be equal to the parameter list length equate (EQU) value that is defined by OSUB_PARMLN.

MBRLIST=*memberlist*
MBRLIST=*(r1-r12)*
Specifies a member list that identifies the command processing clients (such as IMS control regions) in the IMSplex from which to receive unsolicited output messages. Do not specify the MBRLIST= parameter if you specify the TYPELIST parameter. If you do not specify either the TYPELIST or MBRLIST parameter, the default member list is all the command processing clients in the IMSplex.

Use commas to separate the client names.

MBRLISTLEN=*memberlistlen*
MBRLISTLEN=*(r1-r12)*
Specifies the length of the member list that is specified on the MBRLIST parameter.

TYPELIST=*typelist*
TYPELIST=*(r1-r12)*
Specifies a type list that identifies the valid IMSplex member types from which to receive unsolicited output messages. Do not specify the TYPELIST parameter if you specify the MBRLIST parameter. For valid IMSplex member types, see the TYPE= parameter description in "CSLSCREG: registration request" on page 215. If you do not specify either TYPELIST or MBRLIST parameter, the default list is all command processing clients in the IMSplex.

Use commas to separate the client names.

TYPELISTLEN=*typelistlen*
TYPELISTLEN=*(r1-r12)*
Specifies the length of the type list that is specified by the TYPELIST= parameter.

RETNAME=*omname*
RETNAME=*(r1-r12)*
Specifies an 8-byte output field to receive the name of the OM to which the subscription request is sent.

RETTOKEN=*omscitoken*
RETTOKEN=*(r1-r12)*
Specifies a 16-byte output field to receive the OM SCI token that is returned to the caller. This is the SCI token for the target OM address space to which SCI sent the request.

RETCODE=*returncode*
RETCODE=*(r1-r12)*
Specifies a 4-byte field to receive the output return code. OM return codes are defined in the CSLORR. SCI return codes are defined in CSLSRR.

OM or SCI provides the return code. The value of the high-order byte in the return code identifies whether SCI (X'01') or OM (X'02') provided the return code.

RSNCODE=*reasoncode*
RSNCODE=*(r1-r12)*
Specifies a 4-byte field to receive the output reason code. OM return codes are defined in the CSLORR. SCI return codes are defined in the CSLSRR.

SCITOKEN=*scitoken*
SCITOKEN=*(r1-r12)*
Specifies a 16-byte field that contains the SCI token, which uniquely identifies this connection to SCI. The SCI token is returned by a successful CSLSCREG FUNC=REGISTER request.

CSLOMSUB return and reason codes

The return and reason codes shown in the following table can be returned on a CSLOMSUB request.

Table 46. CSLOMSUB request return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'02000014'	X'00005044'	Unable to obtain OSUB block.
	X'00005048'	Unable to obtain storage for member list.
	X'0000504C'	Unable to obtain storage for type list.

Usage notes

If a SPOC subscribes to OM to receive unsolicited output messages, the default is to send all messages to that subscriber. You can prevent IMS, CQS, and CSL messages from being sent as unsolicited output by populating and maintaining a table of messages by using the following macros.

Macro
Description

CSLZUMT
Use this macro to populate table CSLZUMTU with CSL messages that should not be routed to subscribers as unsolicited messages.

CQSUOMT
Use this macro to populate table CQSUOMTU with CQS messages that should not be routed to subscribers as unsolicited messages.

DFSUOMT
Use this macro to populate table DFSUOMTU with IMS messages that should not be routed to subscribers as unsolicited messages.

Samples of the CQSUOMTU, CSLZUMTU, and DFSUOMTU tables are provided in the IMS sample library.

An example of the CSLZUMT macro is shown in the following example. In this example, messages CSL2020I and CSL2021E are added to the CSLZUMTU table, and are therefore prevented from being sent as unsolicited output messages to subscribed clients.

```
CSLZUMT MESSAGE=CSL2020I
CSLZUMT MESSAGE=CSL2021E
```

You can also specify MESSAGE=SUPPRESS in the CQSUOMTU, CSLZUMTU, and DFSUOMTU tables. For example, if the you code the following in message table DFSUOMTU, messages DFSxxxxx and DFSyyyyy will be sent to OM, message DFSzzzzz will not be sent to OM, and any other messages will not be sent to OM because of the MESSAGE=SUPPRESS statement:

```
DFSUOMT MESSAGE=DFSxxxxx,SEND=YES
DFSUOMT MESSAGE=DFSyyyyy,SEND=YES
DFSUOMT MESSAGE=DFSzzzzz,SEND=NO
DFSUOMT MESSAGE=SUPPRESS
```

Another way to control which unsolicited output messages are sent to OM from IMS is to use the new UOM=MTO | NONE | ALL parameter for the DFSDFxxx and DFSCGxxx IMS PROCLIB members.

- With UOM=MTO specified, messages that are destined for the MTO only or system console (or both) only will be sent to OM. Messages being sent to other destinations, such as end user terminals, will not be sent to OM.
- With UOM=NONE specified, no messages will be sent to OM.
- With UOM=ALL specified, all messages will be sent to OM.

CSLOMUSB: Unsubscribe to unsolicited messages

With the CSLOMUSB request, SPOC clients can unsubscribe to OM and stop receiving unsolicited output messages from a command processing client, such as IMS.

This request is supported in Assembler and PLX.

CSLOMUSB syntax

CSLOMUSB FUNC-DSECT syntax

Use the DSECT function of the CSLOMUSB request to include equate (EQU) statements in your logic for the CSLOMUSB parameter list length and return and reason codes.

```
▶▶ CSLOMUSB ── FUNC=DSECT ──▶◀
```

CSLOMUSB request syntax

With this syntax, SPOC clients can unsubscribe from OM to stop receiving unsolicited output messages.

```
▶▶ CSLOMUSB ── FUNC=UNSUBSCRIBE ── PARM=parm ── RETCODE=returncode ──▶
    ▶── RSNCODE=reasoncode ── SCITOKEN=scitoken ──▶◀
```

CSLOMUSB parameters

PARM=*parm*
PARM=*(r1-r12)*
Specifies the CSLOMUSB parameter list. The length of the parameter list must be equal to the parameter list length equate (EQU) value that is defined by OUSB_PARMLN.

RETCODE=*returncode*
RETCODE=*(r1-r12)*
Specifies a 4-byte field to receive the output return code. OM return codes are defined in the CSLORR. SCI return codes are defined in CSLSRR.

OM or SCI provides the return code. The value of the high-order byte in the return code identifies whether SCI (X'01') or OM (X'02') provided the return code.

RSNCODE=*reasoncode*
RSNCODE=*(r1-r12)*
Specifies a 4-byte field to receive the output reason code. OM return codes are defined in the CSLORR. SCI return codes are defined in CSLSRR.

SCITOKEN=*scitoken*
SCITOKEN=*(r1-r12)*
Specifies a 16-byte field that contains the SCI token, which uniquely identifies this connection to SCI. The SCI token is returned by a successful CSLSCREG FUNC=REGISTER request.

CSLOMUSB return and reason code

The return and reason codes shown in the following table can be returned on a CSLOMUSB request:

Table 47. CSLOMUSB request return and reason code

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.

CSL OM directives

An OM directive is a function that OM defines that can be sent as a message to OM clients to inform the OM clients of work to be processed. Any command processing client that has registered commands to OM can be selected to perform an OM directive.

OM directives are always issued in message protocol (PROTOCOL=MSG), that is, asynchronously; OM therefore expects no response from the OM client, and it continues processing without waiting for a response. The OM client is responsible for determining whether or not to take any action in response to the directive.

When a client issues PROTOCOL=MSG, SCI sends the XML output from OM to the client's SCI Input exit. The SCI Input exit routine's INXP_MBRPLPTR field points to the CSLOMDIR parameter list.

When a client issues CSLOMI PROTOCOL=RQST, the XML output stream from OM is sent directly to the client in the OUTPUT= parameter.

The SCI Input exit routine is responsible for notifying the client of the directive. The client should code their SCI Input exit routine to support OM directives. The client is responsible for determining where the function and function code are to be defined. After the client is finished using the CSLOMDIR parameter list, it must release the storage by issuing CSLSCBFR.

OM directives are defined in the CSLOMDIR macro, which includes:

- Command directive (ODIR_CMDD)
- CSLOMI response directive (ODIR_OMIRESPD)
- Command response directive (ODIR_CMDRESPD)
- Query response directive (ODIR_QRYRESPD)

The directives and their parameters are described in this topic.

CSL OM command directive

The OM command directive, ODIR_CMDD, is sent to a command processing client when a command is available for processing.

The parameters for the OM command directive follow. They are passed to the SCI Input Exit:

ODIR_COMMAND
Identifies the start of the command directive.

ODIR_CMDTKLEN=*length*
Contains the length of the OM command token. It is used only by OM to identify the command instance.

ODIR_CMDTKPTR=*address*
Contains the address of the OM command token.

ODIR_INPUTLEN=*length*
Contains the length of the command input string that you enter.

ODIR_INPUTPTR=*address*
Contains the address of the command input string.

ODIR_VERBLEN=*length*
Contains the length of the command verb in normalized form.

ODIR_VERBPTR=*address*
Contains the address of the command verb.

ODIR_KWDLEN=*length*
Contains the length of the command keyword.

ODIR_KWDPTR=*address*
Contains the address of the command keyword.

ODIR_PARSELEN=*length*
Contains the length of the parsed command block.

ODIR_PARSEPTR=*address*
Contains the address of the parsed command block.

ODIR_CUIDLEN=*length*
Contains the length of the user ID that originated the command.

ODIR_CUIDPTR=*address*
Contains the address of the user ID that originated the command.

ODIR_CNAMELEN=*length*
Contains the length of the name of the client that originated the command (that is, the name that was registered to SCI).

ODIR_CNAMEPTR=*address*
Contains the address of the name of the client that originated the command (that is, the name that was registered to SCI).

ODIR_CTYPE=*client type*
Contains the type of client that originated the command. This is the value from the TYPE= parameter as defined to SCI. This parameter is passed by value; the length field is always zero.

ODIR_CSTYPLEN=*length*
Contains the subtype of the client that originated the command. This is the value from the SUBTYPE= parameter as defined to SCI.

ODIR_CSTYPPTR=*length*
Contains the address of the subtype of the client that originated the command.

ODIR_CFLAGS=*flags*
Contains the OM command processing flags. These parameters are passed by value; the length field is always zero.

ODIR_CRQTKLEN=*length*
Contains the length of the user request token; this parameter is used only by the program that originated the command to identify the command instance.

ODIR_CRQTKPTR=*address*
Contains the address of the user request token; this parameter is used only by the program that originated the command to identify the command instance.

ODIR_TIMEOUT=*timeoutvalue*
Contains the command timeout value as specified on the command. This parameter is passed by value; the length field is always zero.

ODIR_CMDLN
The command directive length EQU.

CSL OM response directives

There are three response directives in CSLOMDIR:

- CSLOMI response (ODIR_OMIRESPD)

  The CSLOMI response directive returns a response to a client regarding a CSLOMI call. The response is sent when an error occurs and it is unclear if the response is for a CSLOMI CMD or CSLOMI QUERY call.
- Command response (ODIR_CMDRESPD)

  The command response directive returns a command response to a client that results from a CSLOMI CMD or CSLOMCMD call.
- Query response (ODIR_QRYRESPD)

  The query response directive returns a query response to a client that results from a CSLOMI QUERY or CSLOMQRY call.

The parameters for the OM response directives are identical.

ODIR_CQRESP
Identifies the start of the command or query response.

ODIR_CQRSPRC=*returncode*
Contains the return code of the command or query response.

ODIR_CQRSPRSN=*reasoncode*
Contains the reason code of the command or query response.

ODIR_CQXMLLEN=*length*
Contains the length of the XML output being returned.

ODIR_CQXMLPTR=*address*
Contains the address of the XML output being returned.

ODIR_CQRT1LEN=*length*
Contains the length of request token 1 (RQSTTKN1).

ODIR_CQRT1PTR=*address*
Contains the address of request token 1 (RQSTTKN1).

ODIR_CQRT2LEN=*length*
Contains the length of request token 2 (RQSTTKN2).

ODIR_CQRT2PTR=*address*
Contains the address of request token 2 (RQSTTKN2).

ODIR_CQRSPLN
The response directive length EQU.

CSL UOM directive

The OM unsolicited output message (UOM) directive, ODIR_UOM, is sent to any OM client that subscribes to OM. You define this directive in CSLOMDIR.

The parameters for the UOM directive are passed to the OM client's SCI Input exit routine.

ODIR_UOM
Identifies the start of the UOM directive parameters.

ODIR_UOMXMLLEN=*length*
Contains the length of the XML output that is sent.

ODIR_UOMXMLPTR=*address*
Contains the address of the unsolicited output message, wrapped in XML tags.

ODIR_UOMLN
The UOM directive length equate (EQU) value.

ODIR_UOMCR
Identifies the start of the command response UOM directive parameters, which inform the subscribing client that this is a command response sent as an unsolicited message.

ODIR_UOMXMLLEN=*length*
Contains the length of the XML output that is sent.

ODIR_UOMXMLPTR=*address*
Contains the address of the unsolicited output message, wrapped in XML tags.

ODIR_UOMLN
The UOM directive length equate (EQU) value.

Related reference

“CSLOMI: API request” on page 111
With the CSLOMI request, your AOP client can communicate with a z/OS address space that acts as an OM AOP client. You can then issue OM requests and send QUERY commands to OM.

“CSLOMQRY: query request” on page 121
With the CSLOMQRY request, any AOP client that is running on the host can request OM-specific information.

BPE-based CSL SCI user exit routines (Exit Routines)

Chapter 7. Writing a CSL RM client

The topics in this section describe the client requests and directives for writing an RM client.

Sequence of RM client requests

If you want to use RM to manage global resources in an IMSplex for your own product or service, you have to write one or more RM clients. An RM client uses RM requests issued in a particular sequence to communicate with RM.

To write an RM client, you can use the set of client requests provided by RM. These requests allow a client to access RM or resources on a resource structure, or to coordinate an IMSplex-wide process. One example of an RM client is IMS. You can write an RM client in assembler language.

An RM client uses RM requests to make use of RM services and resources. A client issues SCI and RM requests to request RM services. Some of the requests must follow a particular sequence. Other requests can be issued multiple times, in any order, based on the processing requirements of the client.

Before an RM client can issue RM requests, it must register:

- To SCI
- To each active RM in the IMSplex, so any RM can process an RM request
- Its own resource types and associated name types to RM

The following table lists the sequence of requests issued by an RM client. The request is listed with its purpose.

Table 48. Sequence of requests for an RM client

Request	Purpose
CSLSCREG	Registers to SCI, which enables the client to send RM requests to RM through SCI.
CSLSCRDY	Readies the RM client to SCI, which routes messages to the client by client type.
CSLRMREG	Registers client to RM to enable communication with RM. The client should register to each active RM in the IMSplex, so any RM can process an RM request. The client can also register its own resource types and associated name types to RM.
CSLRMxxx	Issues RM resource requests such as CSLRMUPD, CSLRMDEL, CSLRMQRY to manipulate resources on a resource structure.
CSLRMPxx	Issues RM process requests such as CSLRMPRI, CSLRMPRS, CSLRMPRR, and CSLRMPRT to participate in an IMSplex-wide process.
CSLSCBFR	Releases the output buffer returned by the request, if any.
CSLRMDRG	Deregisters client from RM to end communications with RM.
CSLSCDRG	Deregisters from SCI.

The following table lists the sequence of requests issued by an RM client that is participating in IMSplex-wide processes. The request is listed with its purpose.

Table 49. Sequence of requests for an RM client participating in IMSplex-wide process	
Request	**Purpose**
CSLRMPRI	Initiate an IMSplex-wide process.
CSLRMPRS	Process a step in an IMSplex-wide process. A process can have zero, one, or more process steps. The client that initiates the process step is the master of the step.
CSLRMPRR	Respond to a process step.
CSLRMPRT	Terminate an IMSplex-wide process.

Related reference

“CSLRMREG: register clients” on page 187
You use the CSLRMREG request to register a client to RM and, optionally, to register the client's resource types and associated name types. The client being registered to RM must be authorized to issue a CSLRMREG request. However, you cannot register a client if an IMSplex-wide process is in progress.

Issue CSL RM requests to manage global resources

Before a client can access or change global resource information, it must register to SCI using the CSLSCREG request. After the client registers to SCI, it must register to RM using the CSLRMREG request. The client must issue an SCI registration request for every IMSplex with which it intends to communicate.

Before a client can access or change global resource information, it must register to SCI using the CSLSCREG request.

After the client registers to SCI, it must register to RM using the CSLRMREG request.

When the client is ready to terminate, it must deregister from RM using the CSLRMDRG request and then deregister from SCI using the CSLSCDRG request.

Related reference

“CSLSCREG: registration request” on page 215
The Structured Call Interface (SCI) registration request is used to create a connection between an IMSplex member and SCI. Before SCI can be used for communication within the IMSplex, an IMSplex member must issue the CSLSCREG request and receive an SCI token when the request completes in order to be recognized by SCI.

“CSLRMDEL: delete resources” on page 166
You can issue the CSLRMDEL request to delete one or more uniquely named resources, or all resources by owner for a specific resource type on a resource structure.

“CSLRMDRG: deregister clients” on page 170
The deregister request is issued by a client when the client no longer wants to process resource requests or IMSplex-wide process requests from RM. The deregister request removes client information from RM and stops RM from sending new resource requests to the client. Some information about the client is retained that can affect IMSplex-wide processes.

“CSLRMPRI: process initiate” on page 171
With the CSLRMPRI request, a client can initiate a process across an IMSplex. RM ensures that only one IMSplex-wide process of a type can be in progress at one time. The process initiation fails if any other IMSplex-wide process of the type is in progress.

“CSLRMPRR: process respond” on page 173
By issuing the CSLRMPRR request, a client can respond to a step in an IMSplex-wide process.

“CSLRMPRS: process step” on page 175
By issuing the CSLRMPRS request, a client can perform a step in an IMSplex-wide process that can consist of zero, one, or more steps.

“CSLRMPRT: process terminate” on page 180

You can issue the CSLRMPRT request to terminate an IMSplex-wide process. Any client that is participating in the process can issue a CSLRMPRT FUNC= TERMINATE request to terminate the process.

"CSLRMQRY: query resources" on page 182
You can issue the CSLRMQRY request to query one or more uniquely named resources on a resource structure.

"CSLRMREG: register clients" on page 187
You use the CSLRMREG request to register a client to RM and, optionally, to register the client's resource types and associated name types. The client being registered to RM must be authorized to issue a CSLRMREG request. However, you cannot register a client if an IMSplex-wide process is in progress.

"CSLRMUPD: update resources" on page 190
By issuing the CSLRMUPD request, you can create a resource if it does not exist, or update a resource if it does exist (as long as the version specified matches the version of the resource). A resource can be created or updated with or without client data.

Issue CSL RM requests to coordinate IMSplex-wide processes

You can use RM-supplied requests to coordinate IMSplex-wide processes. All clients that are to participate in the process register to RM using the RM client registration request (CSLRMREG). After the clients are registered, several different requests can be utilized to coordinate processes.

One client initiates the process using the RM process initiate request (CSLRMPRI). The same or a different client initiates a step using RM process step request (CSLRMPRS). The initiating client is called the *master* of the step. One RM processes the request and sends RM directives to the other clients to perform the process step. All the other clients process the step, build output, and then respond to the step using the RM process respond request (CSLRMPRR). RM consolidates the responses from all the clients into one output, and then returns the output to the master of the process step. If there are more steps in the process, a client initiates a step, and the clients perform processing and respond. Any client terminates the process using the RM process terminate request (CSLRMPRT). Clients can deregister using the RM client deregistration request (CSLRMDRG) if required.

Some failures can cause RM to lose all knowledge of an IMSplex-wide process. These include resource structure failure (and its duplex, if applicable) and failure of all RMs. If this type of failure occurs, each RM client should clean up knowledge of the process locally, and a master RM should terminate the process. The first RM client to detect a problem can initiate a clean up process step by issuing the CSLRMPRS request with the force option to enable RM to force the process step, regardless of the error. The clients participating in the process step clean up the process locally. The master of this process step then terminates the process with the CSLRMPRT request.

The CSLRMPRI, CSLRMPRR, CSLRMPRS, and CSLRMPRT requests can be used to coordinate IMSplex-wide processes.

Related reference

"CSLRMPRI: process initiate" on page 171
With the CSLRMPRI request, a client can initiate a process across an IMSplex. RM ensures that only one IMSplex-wide process of a type can be in progress at one time. The process initiation fails if any other IMSplex-wide process of the type is in progress.

"CSLRMPRR: process respond" on page 173
By issuing the CSLRMPRR request, a client can respond to a step in an IMSplex-wide process.

"CSLRMPRS: process step" on page 175
By issuing the CSLRMPRS request, a client can perform a step in an IMSplex-wide process that can consist of zero, one, or more steps.

"CSLRMPRT: process terminate" on page 180

You can issue the CSLRMPRT request to terminate an IMSplex-wide process. Any client that is participating in the process can issue a CSLRMPRT FUNC= TERMINATE request to terminate the process.

CSLRMDEL: delete resources

You can issue the CSLRMDEL request to delete one or more uniquely named resources, or all resources by owner for a specific resource type on a resource structure.

This request is supported in assembler language.

CSLRMDEL syntax

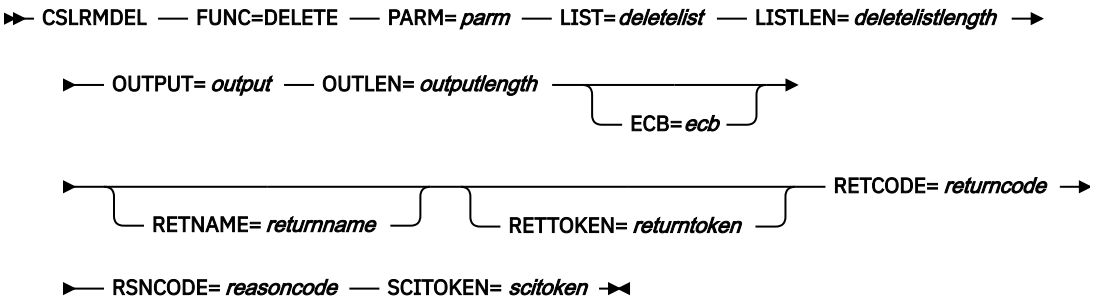
CSLRMDEL DSECT syntax

Use the DSECT function of a CSLRMDEL request to include the following resources in your program:

- Equate (EQU) statement for the CSLRMDEL parameter list length
- The CSLRMDEL return codes, reason codes, and completion codes
- The CSLRDELL DSECT to map the input delete list
- The CSLRDELO DSECT to map the delete output

CSLRMDEL DELETE syntax

Use the DELETE function of a CSLRMDEL request to delete one or more uniquely named resources on a resource structure.



CSLRMDEL parameters

ECB=*symbol*
ECB=*(r2-r12)*
(Optional) - Specifies the address of a z/OS ECB used for asynchronous requests. When the request is complete, the ECB specified is posted. If an ECB is not specified, the task is suspended until the request is complete. If an ECB is specified, the invoker of the request must issue a WAIT (or equivalent) after receiving control from CSLRMDEL before using or examining any data returned by this request (including the RETCODE and RSNCODE fields).

LIST=*symbol*
LIST=*(r2-r12)*
(Required) - Specifies the delete resource list built by the caller. Each list entry is a separate delete request. The list length can vary, depending on the number of list entries.

CSLRDELL maps the delete resource list entry. The list contains a header and one or more list entries. The list entries must reside in contiguous storage. Each delete list entry contains information about what to delete.

For delete by resource name, to delete a uniquely named resource:

- Resource name - the client-defined name of the resource.

- Resource type - a client-defined physical grouping of resources on the resource structure. Valid values are 1-255.
- Version - resource version, which is the number of times the resource has been updated.

For delete by owner, to delete all resources owned by a particular owner for a resource type, regardless of the resource version:

- Resource type - a client-defined physical grouping of the resources on the resource structure. Valid values are 1-255.
- Owner - resource owner.

LISTLEN=*symbol*
LISTLEN=*(r2-r12)*
(Required) - Specifies the 4-byte delete resource list length.

OUTLEN=*symbol*
OUTLEN=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the length of the output returned by the CSLRMDEL request. OUTLEN contains the length of the output pointed to by the OUTPUT= parameter.

The output length is zero if no output is built, for example, if an error is detected before any output can be built.

OUTPUT=*output*
OUTPUT=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the address of the variable length output returned by the CSLRMDEL request. The output contains a header and one or more delete entries for resource deletes that were attempted. The output length is returned in the OUTLEN= field.

The output address is zero if no output was built, for example, if an error was detected before any output could be built.

The CSLRDELO macro maps the output that is returned. The output contains a header and one or more list entries.

The output header contains the following parameters:

- Eyecatcher
- Output length
- CSLRDELO version
- CSLRDELO header length (offset to start of entries)
- CSLRDELO entry length
- Resource entry count

Each output entry represents a resource delete that failed. Each entry contains the following parameters:

- Output entry length - the list entry length
- Name type - a client-defined value associated with a resource type that ensures uniqueness of client-defined resource names within a name type. Valid values are 1-255.
- Resource name
- Resource type
- Delete type
- Version - resource version of an existing resource if the delete request failed because of a version mismatch.
- Owner - resource owner of an existing resource if the delete failed because of a version mismatch and the option to read the owner was set.
- Completion code for the delete request. Completion codes are mapped by CSLRRR.

Possible completion codes are:

X'00000008'
Invalid resource type.

X'00000010'
Version mismatch. The version specified on input does not match the resource's version, so delete fails.

X'00000018'
Resource type is not registered. The resource type must be registered by using a CSLRMREG request.

X'00000024'
Resource structure is unavailable.

X'00000038'
Delete failed because of CQS internal error.

X'0000003C'
Delete failed because RM incorrectly built the CQSDEL list entry.

The output buffer is not preallocated by the caller. After being returned from the request, this word contains the address of a buffer containing the delete output. It is the caller's responsibility to release this storage by issuing the CSLSCBFR FUNC=RELEASE request when it is through with the storage. The length of the output is returned in the OUTLEN= field.

PARM=*symbol*
PARM=*(r2-r12)*
(Required) - specifies the CSLRMDEL parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by RDEL_PARMLN.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - specifies a 4-byte field to receive the return code on output. RM return codes are defined in CSLRRR. SCI return codes are defined in CSLSRR. Possible return codes are described in the following table.

RETNAME=*symbol*
RETNAME=*(r2-r12)*
(Optional) - Specifies an 8-byte field to receive the RM name returned to the caller. This is the CSL member name of the target RM address space to which SCI sent the request.

RETTOKEN=*symbol*
RETTOKEN=*(r2-r12)*
(Optional) - Specifies a 16-byte field to receive RM's SCI token returned to the caller. This is the SCI token for the target RM address space to which SCI sent the request.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. RM reason codes are defined in CSLRRR. SCI reason codes are defined in CSLSRR. Possible reason codes are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

CSLRMDEL return and reason codes

The following table lists the return and reason codes that can be returned on a CSLRMDEL request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 50. CSLRMDEL return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'03000008'	X'00002000'	The client is not registered.
	X'00002100'	The delete list length is invalid.
	X'00002108'	The delete list address is invalid.
	X'00002110'	The version in the list header (DELL_PVER) is zero, which is invalid. The list version must be set in the list header to the maximum list version (DELL_PVERMAX).
	X'00002114'	The list header length is invalid. The list header length cannot be zero or greater than the list length that was passed in. The list header length (DELL_HDRLEN) must be set in the list header to the list header length.
	X'00002200'	One of the list entries contains an invalid resource type, such as zero. RM assumes that the rest of the list is invalid.
	X'0000220C'	One of the list entries contains one or more invalid delete options. RM assumes that the rest of the list is invalid.
	X'00002210'	A resource name or owner is required.
	X'00002214'	The version is invalid.
	X'00002404'	No resource structure is defined.
X'0300000C'	X'00003000'	The request succeeded for at least one, but not all, list entries. Check the completion code in each list entry in the OUTPUT buffer for individual errors.
	X'00003004'	The request failed for all entries. Check the completion code in each list entry in the OUTPUT buffer for individual errors.
	X'00003008'	The request failed for one or more list entries and all failures were version mismatches. Check the completion code in each list entry in the OUTPUT buffer for individual errors.
X'03000010'	X'00004000'	The CQS address space is unavailable. Retry the request again, which attempts to route the request to a different RM with an available CQS.
	X'00004100'	The requested version is not supported. The client compiled with a version of CSLRMDEL that is not supported by RM. All RMs must be migrated to a new release before IMS is migrated to a new release that uses a new CSLRMDEL function.
	X'00004104'	The list version is not supported. The client created the delete list at a version that is not supported by RM. All RMs must be migrated to a new release before the client is migrated to a new release that uses a new CSLRMDEL function.
X'03000014'	X'00005000'	Storage allocation for the delete output buffer failed.
	X'00005120'	Storage allocation for the CQSDEL buffer failed.
	X'00005200'	The CQS request resulted in unexpected error.
	X'00005204'	The CQS request failed because RM incorrectly built the request input.

Related concepts

"Issue CSL RM requests to manage global resources" on page 164
Before a client can access or change global resource information, it must register to SCI using the CSLSCREG request. After the client registers to SCI, it must register to RM using the CSLRMREG request. The client must issue an SCI registration request for every IMSplex with which it intends to communicate.

CSLRMDRG: deregister clients

The deregister request is issued by a client when the client no longer wants to process resource requests or IMSplex-wide process requests from RM. The deregister request removes client information from RM and stops RM from sending new resource requests to the client. Some information about the client is retained that can affect IMSplex-wide processes.

This request is issued by resource processing clients such as the IMS control region.

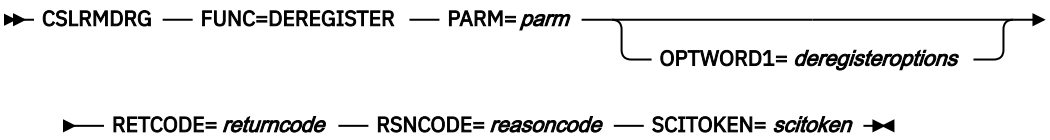
This request is supported in assembler.

CSLRMDRG syntax

CSLRMDRG DSECT syntax

CSLRMDRG deregister syntax

Use the DEREGISTER function of a CSLRMDRG request to deregister from RM.



CSLRMDRG parameters

OPTWORD1=*symbol*
OPTWORD1=*(r2-r12)*
(Optional) - Specifies a 4-byte field containing deregistration options. CSLRMDRG FUNC=DSECT generates the equates for deregistration options.

X'80000000'
Remove client from IMSplex. Delete all knowledge of the client.

PARM=*symbol*
PARM=*(r2-r12)*
(Required) - Specifies the CSLRMDRG parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by RDRG_PARMLN.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the return code on output. RM return codes are defined in CSLRRR. RM does not return a response to the CSLRMDRG request.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. SCI reason codes are defined in CSLSRR. RM does not return a response to the CSLRMDRG request.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

Related concepts

"Issue CSL RM requests to manage global resources" on page 164

Before a client can access or change global resource information, it must register to SCI using the CSLSCREG request. After the client registers to SCI, it must register to RM using the CSLRMREG request. The client must issue an SCI registration request for every IMSplex with which it intends to communicate.

CSLRMPRI: process initiate

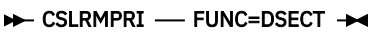
With the CSLRMPRI request, a client can initiate a process across an IMSplex. RM ensures that only one IMSplex-wide process of a type can be in progress at one time. The process initiation fails if any other IMSplex-wide process of the type is in progress.

This request is supported in assembler language.

CSLRMPRI syntax

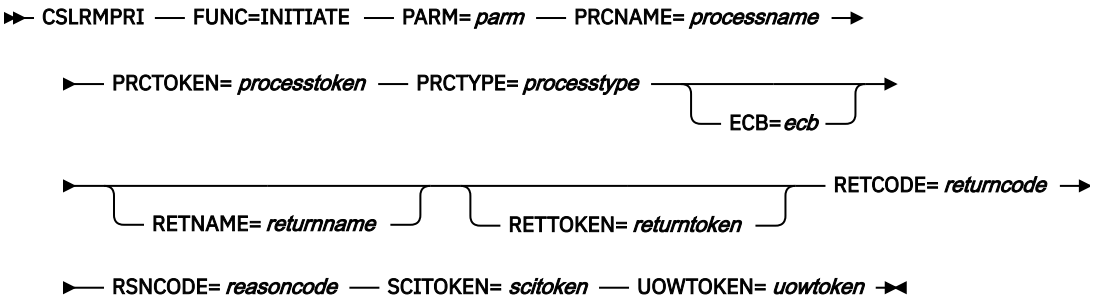
CSLRMPRI DSECT syntax

Use the DSECT function of a CSLRMPRI request to include equate (EQU) statements in your program for the length of the CSLRMPRI parameter list.



CSLRMPRI INITIATE syntax

Use the INITIATE function of a CSLRMPRI request to initiate an IMSplex-wide process.



CSLRMPRI parameters

ECB=*symbol*
ECB=*(r2-r12)*
(Optional) - Specifies the address of a z/OS ECB used for asynchronous requests. When the request is complete, the ECB specified is posted. If an ECB is not specified, the task is suspended until the request is complete. If an ECB is specified, the invoker of the request must issue a WAIT (or equivalent) after receiving control from CSLRMPRI before using or examining any data returned by this request (including the RETCODE and RSNCODE fields).

PARM=*symbol*
PARM=*(r2-r12)*
(Required) - Specifies the CSLRMPRI parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by RPRI_PARMLN.

PRCNAME=*symbol*
PRCNAME=*(r2-r12)*
(Required) - Specifies an 8-byte field containing the process name. The process name is client defined and has no meaning to RM. RM uses the process name and the process type to ensure that only one instance of a process of a particular process type is in progress at one time.

PRCTOKEN=*symbol*
PRCTOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field to receive the process token returned to the caller. The process token uniquely identifies the process instance. The process token returned is zero, if the IMSplex is

defined with a resource structure. The process token is non-zero, if the IMSplex is not defined with a resource structure. The process token must be specified as input on any subsequent CSLRMPRS, CSLRMPRR, or CSLRMPRT request.

PRCTYPE=*symbol*
PRCTYPE=*(r2-r12)*
(Required) - Specifies a 1-byte client-defined process type. Only one process of a particular type can be in progress at any one time. The process type can be 1 through 255.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the return code on output. RM return codes are defined in CSLRRR. SCI return codes are defined in CSLSRR. Possible return codes are described in the following table.

RETNAME=*symbol*
RETNAME=*(r2-r12)*
(Optional) - Specifies an 8-byte field to receive the RM name returned to the caller. This is the CSL member name of the target RM address space to which SCI sent the request.

RETTOKEN=*symbol*
RETTOKEN=*(r2-r12)*
(Optional) - Specifies a 16-byte field to receive RM's SCI token returned to the caller. This is the SCI token for the target RM address space to which SCI sent the request.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. RM reason codes are defined in CSLRRR. SCI reason codes are defined in CSLSRR. Possible reason codes are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

UOWTOKEN=*symbol*
UOWTOKEN=*(r2-r12)*
(Required) - specifies a 16-byte field containing the unit of work token. The UOW token uniquely identifies an instance of this process and ties all of the process steps together. The UOW token must be specified on the RM process step request, CSLRMPRS. The UOW token is client-defined and has no meaning to RM.

CSLRMPRI return and reason codes

The following table lists the return and reason codes that can be returned on a CSLRMPRI request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 51. CSLRMPRI return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'03000008'	X'00002000'	The client is not registered.
	X'00002208'	The process type is invalid.
	X'00002310'	The UOW token is invalid.
X'03000010'	X'00004000'	The CQS address space is unavailable. Retry the request to attempt routing the request to another RM with an available CQS.

Table 51. CSLRMPRI return and reason codes (continued)

Return code	Reason code	Meaning
	X'00004100'	The requested version is not supported. The client compiled with a version of CSLRMPRI that is not supported by RM. All RMs must be migrated to a new release before IMS is migrated to a new release that uses a new CSLRMPRI function.
	X'00004120'	A process of the same type is already in progress. This process initiation request is rejected. Try the process again later.
	X'0000412C'	A different process of the same type is already in progress. This process initiation request is rejected. Try the process again later.
X'03000014'	X'00005114'	The process block allocation failed.
	X'00005200'	The CQS request resulted in unexpected error.
	X'00005204'	The CQS request failed because RM incorrectly built the request input.
	X'00005208'	The resource structure is not available.
	X'0000520C'	The resource structure is full.
	X'00005210'	RM is unable to add the process block to hash table.
	X'00005218'	RM is unable to scan the process block in hash table.
	X'00005220'	RM is unable to get the process latch.

Related concepts

"Issue CSL RM requests to manage global resources" on page 164
Before a client can access or change global resource information, it must register to SCI using the CSLSCREG request. After the client registers to SCI, it must register to RM using the CSLRMREG request. The client must issue an SCI registration request for every IMSplex with which it intends to communicate.

"Issue CSL RM requests to coordinate IMSplex-wide processes" on page 165
You can use RM-supplied requests to coordinate IMSplex-wide processes. All clients that are to participate in the process register to RM using the RM client registration request (CSLRMREG). After the clients are registered, several different requests can be utilized to coordinate processes.

CSLRMPRR: process respond

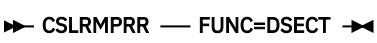
By issuing the CSLRMPRR request, a client can respond to a step in an IMSplex-wide process.

This request is supported in assembler language.

CSLRMPRR syntax

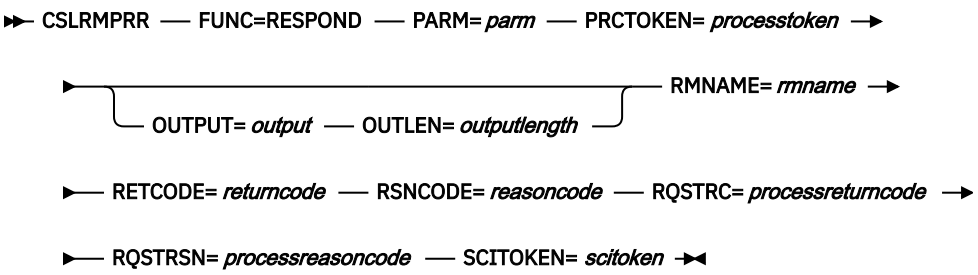
CSLRMPRR DSECT syntax

Use the DSECT function of a CSLRMPRR request to include equate (EQU) statements in your program for the length of the CSLRMPRR parameter list.



CSLRMPRR RESPOND syntax

Use the RESPOND function of a CSLRMPRR request to respond to a step in an IMSplex-wide process.



CSLRMPRR parameters

OUTLEN=*symbol*
OUTLEN=*(r2-r12)*
(Required) - Specifies a 4-byte input field that contains the length of the process step output buffer. OUTLEN= contains the length of the output pointed to by the OUTPUT= parameter.

OUTPUT=*output*
OUTPUT=*(r2-r12)*
(Required) - Specifies a 4-byte field that contains the address of the output buffer built by the caller. The output is client-defined and contains the results from this client's processing of the step. The output length is returned in the OUTLEN= field.

PARM=*symbol*
PARM=*(r2-r12)*
(Required) - Specifies the CSLRMPRR parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by RPRR_PARMLN.

PRCTOKEN=*symbol*
PRCTOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field that contains the process token that uniquely identifies the process. This token was returned on a successful CSLRMPRI FUNC=INITIATE request.

If the IMSplex is defined with a resource structure, the process token is zero.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the return code on output. SCI return codes are defined in CSLSRR. RM does not return a response to CSLRMPRR.

RMNAME=*symbol*
RMNAME=*(r2-r12)*
(Required) - Specifies an 8-byte field containing the RM name to which to send the process step response. This is the RM that originated the process step.

RQSTRC=*symbol*
RQSTRC=*(r2-r12)*
(Required) - Specifies a 4-byte field that contains the return code to be passed to the originator of the process step on output. The return code is defined by the process step originating client and indicates the result of the process step.

RQSTRSN=*symbol*
RQSTRSN=*(r2-r12)*
(Required) - Specifies a 4-byte field that contains the reason code to be passed to the originator of the process step on output. The reason code is defined by the process step originating client and indicates the result of the process step.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. RM reason codes are defined in CSLRRR. RM does not return a response to CSLRMPRR.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

CSLRMPRR return and reason codes

CSLRMPRR is sent to the target client address space using the SCI message protocol; RM does not return codes to CSLRMPRR. All return and reason codes that are applicable to the CSLSCMSG request can be returned on a CSLRMPRR request.

Related concepts

"Issue CSL RM requests to manage global resources" on page 164
Before a client can access or change global resource information, it must register to SCI using the CSLSCREG request. After the client registers to SCI, it must register to RM using the CSLRMREG request. The client must issue an SCI registration request for every IMSplex with which it intends to communicate.

"Issue CSL RM requests to coordinate IMSplex-wide processes" on page 165
You can use RM-supplied requests to coordinate IMSplex-wide processes. All clients that are to participate in the process register to RM using the RM client registration request (CSLRMREG). After the clients are registered, several different requests can be utilized to coordinate processes.

CSLRMPRS: process step

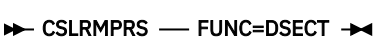
By issuing the CSLRMPRS request, a client can perform a step in an IMSplex-wide process that can consist of zero, one, or more steps.

This request is supported in assembler language.

CSLRMPRS syntax

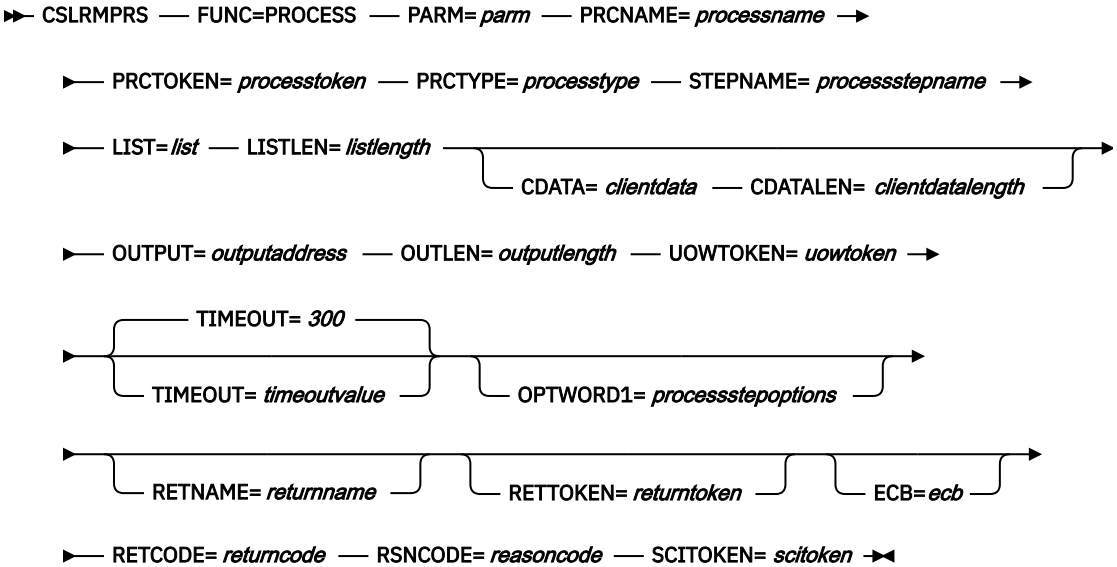
CSLRMPRS DSECT syntax

Use the DSECT function of a CSLRMPRS request to include equate (EQU) statements in your program for the length of the CSLRMPRS parameter list and the process step request options.



CSLRMPRS PROCESS syntax

Use the PROCESS function of a CSLRMPRS request to perform a step in an IMSplex-wide process.



CSLRMPRS parameters

CDATA=*symbol*
CDATA=*(r2-r12)*
(Optional) - Specifies a variable length area that contains client data to send to clients participating in the IMSplex-wide process step. The client data has meaning to clients, not to RM.

CDATALEN=*symbol*
CDATALEN=*(r2-r12)*
(Optional) - Specifies a 4-byte input field that contains the client data length. If this parameter is specified, CDATA= must also be specified.

ECB=*symbol*
ECB=*(r2-r12)*
(Optional) - Specifies the address of a z/OS ECB used for asynchronous requests. When the request is complete, the ECB specified is posted. If an ECB is not specified, the task is suspended until the request is complete. If an ECB is specified, the invoker of the request must issue a WAIT (or equivalent) after receiving control from CSLRM PRS before using or examining any data returned by this request (including the RETCODE and RSNCODE fields).

LIST=*symbol*
LIST=*(r2-r12)*
(Required) - Specifies the variable length input list that contains the list of clients to which to send the process step.

The process step list contains a list header and one or more list entries. The list header contains the list header length, the parameter list version, the list entry length, the list entry count, and a user data area. The list header user data area is passed back to the requestor in the list header of the process step output. Each list entry contains the client name and an optional user data area. The user data area is passed back to the requestor in a list entry in the process step output. The list entries must reside in contiguous storage.

The CSLRPRSL macro maps the process step list.

LISTLEN=*symbol*
LISTLEN=*(r2-r12)*
(Required) - Specifies a 4-byte input field that contains the process step list length.

OPTWORD1=*symbol*
OPTWORD1=*(r2-r12)*
(Optional) - Specifies a 4-byte field containing the process step options. CSLRMPRS FUNC=DSECT maps the process step options.

X'80000000'
Force process step after error. Take over a process step in progress, if a process step is already in progress for an IMSplex member that is not active. Initiate a process and perform a process step if no process is known to be in progress due to an error such as resource structure failure.

OUTLEN=*symbol*
OUTLEN=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the length of the output buffer returned by the CSLRMPRS request. After being returned by request, this word contains the length of the buffer pointed to by the OUTPUT= parameter. If no output is built, the output buffer length is zero. This can occur if an error is detected before any output can be built.

It is the caller's responsibility to release this storage by issuing the CSLSCBFR FUNC=RELEASE request when it is through with the storage.

OUTPUT=*outputaddress*
OUTPUT=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the address of the variable length output buffer returned by the CSLRMPRS request. The output buffer contains the client-defined data from each participating client and indicates the results of the process step. The output buffer length is returned in the OUTLEN= field.

If no output is built, the output buffer address is zero. This can occur if an error is detected before any output can be built.

The CSLRPRSO macro maps the output buffer that is returned. The output buffer header contains an eyecatcher, the output buffer length, the CSLRPRSO version, the header length (offset to start of the process list entries), the list entry minimum size, the process list entry count, a user data area, and the CSLRPRSO create time stamp. The user data area contains the user data passed in the input process step list header.

Each output buffer entry represents the results from a client that participated in a process step. Each entry contains the following:

- Entry length
- Client name
- User data - the user data passed in the input process step list
- Process step response length
- Process step response
- Completion code (CSLRRR) - possible completion codes are:

X'00000000'
Client processes step successfully.

X'00000044'
Client did not respond before the process step timed out.

X'00000048'
The client was not sent the process step request because the client is not registered to RM.

This buffer is not preallocated by the caller. After the request returns it, this word contains the address of a buffer containing information from the IMSplex members participating in the process. It is the caller's responsibility to release this storage by issuing the CSLSCBFR FUNC=RELEASE request when it is through with the storage.

PARM=*symbol*
PARM=*(r2-r12)*
(Required) - Specifies the CSLRMPRS parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by RPRS_PARMLN.

PRCNAME=*symbol*
PRCNAME=*(r2-r12)*
(Required) - Specifies an 8-byte field containing the process name. The process name is client defined and has no meaning to RM. RM uses the process name and type to ensure that only one instance of a process, with a particular process type, is in progress at one time.

PRCTOKEN=*symbol*
PRCTOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field that contains the process token that uniquely identifies the process. This token was returned on a successful CSLRMPRI FUNC=INITIATE request.

If the IMSplex is defined with a resource structure, the process token is zero.

PRCTYPE=*symbol*
PRCTYPE=*(r2-r12)*
(Required) - Specifies a 1-byte client-defined process type. Only one process of a particular type can be in progress at any one time. The process type can be 1 through 255.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the return code on output. SCI return codes are defined in CSLSRR. SCI return codes are defined in CSLSRR. Possible return codes are described in the following table.

RETNAME=*symbol*
RETNAME=*(r2-r12)*
(Optional) - Specifies an 8-byte field to receive the RM name returned to the caller. This is the CSL member name of the target RM address space to which SCI sent the request.

RETNAME=*symbol*
RETNAME=*(r2-r12)*
(Optional) - Specifies an 8-byte field to receive the RM name returned to the caller. This is the CSL member name of the target RM address space to which SCI sent the request.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. RM reason codes are defined in CSLRRR. SCI reason codes are defined in CSLSRR. Possible reason codes are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

STEPNAME=*symbol*
STEPNAME=*(r2-r12)*
(Required) - Specifies an 4-byte field containing the process step name. The process step name is client-defined and has no meaning to RM. Each process step must have a different name.

TIMEOUT=*timeoutvalue*
TIMEOUT=*symbol*
TIMEOUT=*(r2-r12)*
(Optional) - Specifies a 4-byte field containing the process step timeout value in seconds. If the timeout value is reached during the processing of the step, before all of the participants have responded to the process step, RM terminates the process step and returns the available responses. If the specified timeout value is too small, an incomplete response is returned. The TIMEOUT value ensures a response is returned even if a client processing the step is unable to respond.

The default timeout value is 5 minutes (300 seconds). Specify a negative one (-1) value if no timeout is required for the request.

The TIMEOUT value is the shortest possible time value that can cause the process step to time out. RM internally sets a timer to pop every 5 seconds. When the RM timer pops, RM checks to see if any process step timeout value has expired. When the process step timeout value is less than the RM timer value, the actual length of step processing can be longer than the user specified TIMEOUT value.

UOWTOKEN=*symbol*
UOWTOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the unit of work token. The UOW token uniquely identifies an instance of this process and ties all of the process steps together. The UOW token must match the UOW token specified on the CSLRMPRI FUNC=INITIATE request. The UOW token is client-defined and has no meaning to RM.

CSLRMPRS return and reason codes

The following table lists the return and reason codes that can be returned on a CSLRMPRS request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 52. CSLRMPRS return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'03000008'	X'00002000'	The client is not registered.

Table 52. CSLRMPRS return and reason codes (continued)

Return code	Reason code	Meaning
	X'00002110'	The list version in the list header (PRSL_PVER) is zero, which is invalid. The list version must be set in the list header to the maximum list version (PRSL_PVERMAX).
	X'00002114'	The list header length cannot be zero or greater than the list length that was passed in. The list header length (PRSL_HDRLEN) must be set in the list header to the list header length.
	X'00002140'	The client data length cannot be zero or greater than 256.
	X'00002208'	The process type is invalid.
	X'0000220C'	The process step options are invalid.
	X'00002300'	The process token is invalid.
	X'00002310'	The UOW token is invalid.
X'0300000C'	X'00003000'	The process step succeeded for at least one client, but not all. Check the completion code in each list entry in the OUTPUT buffer for individual errors.
	X'00003004'	The request failed for all clients. Check the completion code in each list entry in the OUTPUT buffer for individual errors.
X'03000010'	X'00004000'	The CQS address space is unavailable. Retry the request to attempt routing the request to another RM with an available CQS.
	X'00004100'	The requested version is not supported. The client compiled with a version of CSLRMPRS that is not supported by RM. All RMs must be migrated to a new release before IMS is migrated to a new release that uses a new CSLRMPRS function.
	X'00004104'	The version of the list is not supported. The client created the process step list at a version that is not supported by RM. All RMs must be migrated to a new release before the client is migrated to a new release that uses a new CSLRMPRS function.
	X'00004108'	The SCI address space is unavailable. SCI was available to send the CSLRMPRS request to RM. RM tried coordinating the process step by sending SCI messages to the active clients. The SCI request to send a message to SCI failed for one or more active clients that did not have an SCI active on the system. Some of the clients might have successfully processed the step.
	X'00004124'	A process is not in progress. The process step is rejected.
	X'00004128'	A process step is already in progress. The process step is rejected. If a process step is already in progress because an error occurred while a previous process step was in progress, and the owner of that process step is still active, the next process step must be specified by the owner of the process step with the FORCE option.
X'03000014'	X'00005000'	Storage allocation for the output buffer failed. The process step might or might not have succeeded.
	X'00005114'	The process block allocation failed.

Table 52. CSLRMPRS return and reason codes (continued)

Return code	Reason code	Meaning
	X'00005118'	The process step response block allocation failed.
	X'00005200'	The CQS request resulted in an unexpected error.
	X'00005204'	The CQS request failed because RM incorrectly built the request input.
	X'00005208'	The resource structure is not available.
	X'00005210'	RM is unable to add the process block to hash table.
	X'00005214'	RM is unable to find the process block in hash table.
	X'00005218'	RM is unable to scan the process block in hash table.
	X'00005300'	An SCI error was encountered. SCI was available to send the CSLRMPRS request to RM. RM tried coordinating the process step by sending SCI messages to the active clients. The SCI request to send a message to SCI failed with an error for one or more active clients. Some of the clients might have successfully processed the step.

Related concepts

"Issue CSL RM requests to manage global resources" on page 164
Before a client can access or change global resource information, it must register to SCI using the CSLSCREG request. After the client registers to SCI, it must register to RM using the CSLRMREG request. The client must issue an SCI registration request for every IMSplex with which it intends to communicate.

"Issue CSL RM requests to coordinate IMSplex-wide processes" on page 165
You can use RM-supplied requests to coordinate IMSplex-wide processes. All clients that are to participate in the process register to RM using the RM client registration request (CSLRMREG). After the clients are registered, several different requests can be utilized to coordinate processes.

CSLRMPRT: process terminate

You can issue the CSLRMPRT request to terminate an IMSplex-wide process. Any client that is participating in the process can issue a CSLRMPRT FUNC= TERMINATE request to terminate the process.

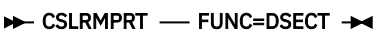
This request is supported in assembler language.

CSLRMPRT syntax

The syntax for the CSLRMPRT request follows.

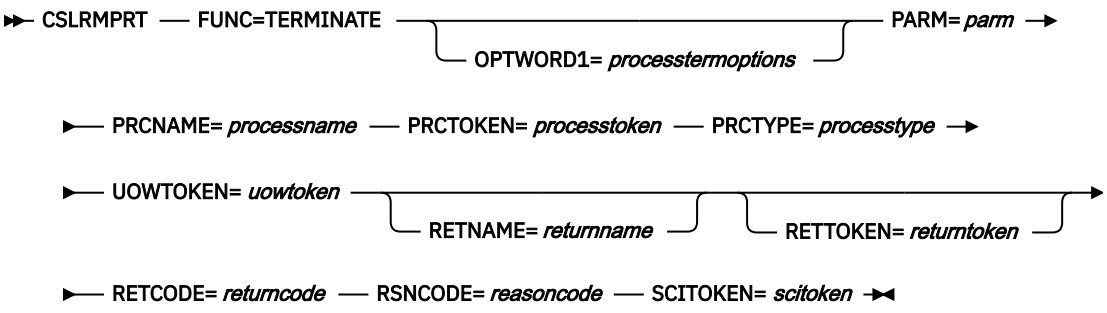
DSECT syntax

Use the DSECT function of a CSLRMPRT request to include equate (EQU) statements in your program for the length of the CSLRMPRT parameter list.



TERMINATE syntax

Use the TERMINATE function of a CSLRMPRT request to terminate an IMSplex-wide process.



CSLRMPRT parameters

OPTWORD1=*symbol*
OPTWORD1=*(r2-r12)*

(Optional) - Specifies a 4-byte field containing the process terminate options. CSLRMPRT FUNC=DSECT maps the process terminate options.

The following options can be set in OPTWORD1:

- X'80000000': Force process to terminate
- X'40000000': Suppress process to terminate error messages

PARM=*symbol*
PARM=*(r2-r12)*

(Required) - Specifies the CSLRMPRT parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by RPRT_PARMLN.

PRCNAME=*symbol*
PRCNAME=*(r2-r12)*

(Required) - Specifies an 8-byte field containing the process name. The process name is client defined and has no meaning to RM. RM uses the process name and the process type to ensure that only one instance of a process of a particular process type is in progress at one time.

PRCTOKEN=*symbol*
PRCTOKEN=*(r2-r12)*

(Required) - Specifies a 16-byte field that contains the process token that uniquely identifies the process. This token was returned on a successful CSLRMPRI FUNC=INITIATE request.

If the IMSplex is defined with a resource structure, the process token is zero.

PRCTYPE=*symbol*
PRCTYPE=*(r2-r12)*

(Required) - Specifies a 1-byte client-defined process type. Only one process of a particular type can be in progress at any one time. The process type can be 1 through 255.

RETCODE=*symbol*
RETCODE=*(r2-r12)*

(Required) - Specifies a 4-byte field to receive the return code on output. SCI return codes are defined in CSLSRR. RM does not return a response to CSLRMPRT.

RETNAME=*symbol*
RETNAME=*(r2-r12)*

(Optional) - Specifies an 8-byte field to receive the name of the RM address space to which SCI sent the process terminate request.

RETTOKEN=*symbol*
RETTOKEN=*(r2-r12)*

(Optional) - Specifies a 16-byte field to receive the SCI token of the RM address space to which SCI sent the process terminate request.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. SCI reason codes are defined in CSLSRR. RM does not return a response to the CSLRMPRT request.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

UOWTOKEN=*symbol*
UOWTOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the unit of work token. The UOW token uniquely identifies an instance of this process and ties all of the process steps together. The UOW token must match the UOW token specified on the CSLRMPRI FUNC=INITIATE request. The UOW token is client-defined and has no meaning to RM.

CSLRMPRT return and reason codes

CSLRMPRT is sent to the target client address space using the SCI message protocol. All return and reason codes that are applicable to the CSLSCMSG request can be returned on a CSLRMPRT request. CSLRMPRT does not issue any additional return and reason codes.

Related concepts

"Issue CSL RM requests to manage global resources" on page 164
Before a client can access or change global resource information, it must register to SCI using the CSLSCREG request. After the client registers to SCI, it must register to RM using the CSLRMREG request. The client must issue an SCI registration request for every IMSplex with which it intends to communicate.

"Issue CSL RM requests to coordinate IMSplex-wide processes" on page 165
You can use RM-supplied requests to coordinate IMSplex-wide processes. All clients that are to participate in the process register to RM using the RM client registration request (CSLRMREG). After the clients are registered, several different requests can be utilized to coordinate processes.

CSLRMQRY: query resources

You can issue the CSLRMQRY request to query one or more uniquely named resources on a resource structure.

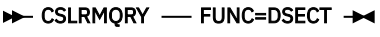
This request is supported in assembler language.

CSLRMQRY syntax

CSLRMQRY DSECT syntax

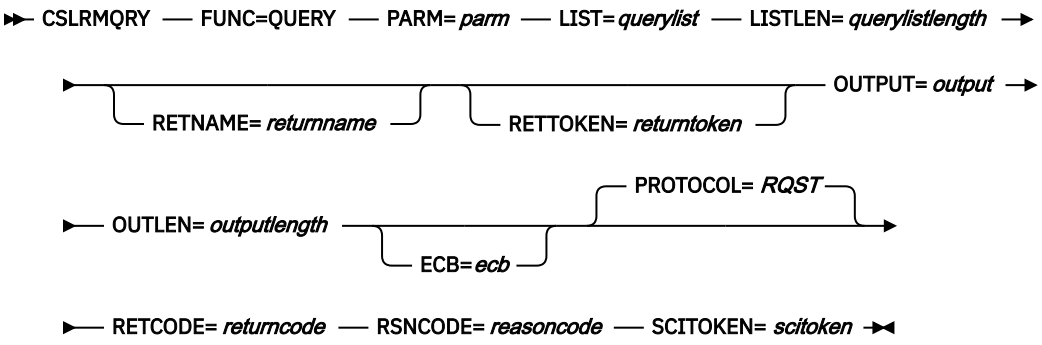
Use the DSECT function of a CSLRMQRY request to include the following inputs and outputs in your program:

- Equate (EQU) statements for the length of the CSLRMQRY parameter list
- The CSLRMQRY return codes, reason codes, and completion codes
- The CSLRQRYL DSECT to map the input query list
- The CSLRQRYO DSECT to map the query output



CSLRMQRY QUERY syntax

Use the QUERY function of a CSLRMQRY request to query one or more uniquely named resources on a resource structure.



CSLRMQRY parameters

ECB=*symbol*
ECB=*(r2-r12)*
(Optional) - Specifies the address of a z/OS ECB used for asynchronous requests. When the request is complete, the ECB specified is posted. If an ECB is not specified, the task is suspended until the request is complete. If an ECB is specified, the invoker of the request must issue a WAIT (or equivalent) after receiving control from CSLRM QRY before using or examining any data returned by this request (including the RETCODE and RSNCODE fields).

LIST=*symbol*
LIST=*(r2-r12)*
(Required) - Specifies the query resource list built by the caller. Each list entry is a separate query request. The list length can vary, depending upon the number of list entries.

The list contains a header and one or more list entries. The list entries must reside in contiguous storage. Each query list entry contains the following:

- Resource name - the client-defined name of the resource. The resource name can be a wildcard name. If it is a wildcard name, all resources that match the wildcard name are returned.
- Resource type - the resource type is a client-defined physical grouping of resources on the resource structure. Valid values are 1-255.
- Query options (optional) - options that indicate special processing to perform for the query.
- Owner (optional) - the owner of the resource. If you specify the owner, the resource is returned only if the resource name and owner match a resource on the resource structure. Specify binary zeroes to omit the owner, and the query returns the owner name in the RQYO_OWNER field in the output list entry.
- User (optional) - a user field set by the caller, which is passed back in the output list entry associated with the input list entry.

LISTLEN=*symbol*
LISTLEN=*(r2-r12)*
(Required) - Specifies the 4-byte query resource list length.

OUTLEN=*symbol*
OUTLEN=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the length of the output buffer returned by the CSLRMQRY request. OUTLEN contains the length of the output buffer pointed to by the OUTPUT parameter. The length of the output data (header and entries) is passed in the output header data, mapped by CSLRQRYO.

OUTPUT=*output*
OUTPUT=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the address of the variable length output returned by the CSLRMQRY request. The output contains a header and one or more query entries for resource queries that were attempted. The output length is returned in the OUTLEN field.

The output address is zero if no output was built, for example, if an error was detected before any output could be built.

The CSLRQRYO macro maps the output that is returned. The output contains a header and one or more list entries. The header contains the following:

- an eyecatcher
- the output length
- CSLRQRYO version
- CSLRQRYO header length (offset to start of entries)
- minimum entry length (offset to DATA2)
- resource entry count
- time stamp

Each output entry represents a resource query that was attempted. Each entry contains the following parameters:

- Output entry length - the list entry length can vary, depending upon whether DATA2 is returned.
- Name type - the name type is a client-defined value associated with a resource type that ensures uniqueness of client-defined resource names within a name type. Valid values are 1-255.
- Resource name - client-defined name of the resource.
- Resource type - the resource type is a client-defined physical grouping of resources on the resource structure. Valid values are 1-255.
- Version - the resource version, which is the number of times the resource has been updated.
- DATA2 flag byte - flag byte indicating if DATA2 was read.
- Resource name status flag - the resource name status indicates how the resource name in the query output list entry is associated with the input resource parameter. This enables you to tie the input resource parameter to the output query list entries that are generated. The following resource name status are possible:

  Specific parameter
  A specific resource name was specified. This query list entry contains the resource name that matches the input parameter.

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

  Wildcard match
  A wildcard parameter was specified. This entry contains information about one resource that matches the input wildcard parameter. All wildcard match list entries follow contiguously after a wildcard parameter list entry.
- Owner - owner of a resource.
- DATA1- a small piece of client data (fixed length, contained in the adjunct area of a data entry) associated with an existing resource.
- DATA2 length - length of a large piece of client data associated with an existing resource, if DATA2 exists and the option to read DATA2 was set.
- Optional User field - optional 4 byte user field passed back to the caller in the output list entry associated with the input list entry.
- DATA2 - a large piece of client data (variable length, contained in one or more data elements of a data entry) associated with an existing resource, if DATA2 exists, and the option to read DATA2 was set. The maximum size of DATA2 is 61312 bytes (X'EF80').
- Completion code for the query request - completion codes are mapped by CSLRRR. Possible completion codes are:

X'00000000'
Query request succeeded. At least one resource matching the query parameters is returned in the output buffer specified by OUTPUT=.

X'00000004'
No resources found.

X'00000008'
Invalid resource type.

X'0000000C'
Invalid name type.

X'00000024'
Resource structure is unavailable.

X'00000034'
Invalid options specified.

X'00000038'
Query failed because of CQS internal error.

X'0000003C'
Query failed because RM incorrectly built the CQSBRWSE list entry.

The output buffer is not preallocated by the caller. After the request returns it, this word contains the address of a buffer containing the query output. It is the caller's responsibility to release this storage by issuing the CSLSCBFR FUNC=RELEASE request when it is through with the storage. The length of the buffer is returned in the OUTLEN= field.

PARM=*symbol*
PARM=*(r2-r12)*
(Required) - Specifies the CSLRMQRY parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by RQRY_PARMLN.

PROTOCOL=RQST
(Optional) - SCI protocol for sending the request to RM. RQST sends the query request using SCI request interface.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - specifies a 4-byte field to receive the return code on output. RM return codes are defined in CSLRRR. SCI return codes are defined in CSLSRR. Possible return codes are described in the following table.

RETNAME=*symbol*
RETNAME=*(r2-r12)*
(Optional) - Specifies an 8-byte field to receive the RM name returned to the caller. This is the CSL member name of the target RM address space to which SCI sent the request.

RETTOKEN=*symbol*
RETTOKEN=*(r2-r12)*
(Optional) - Specifies a 16-byte field to receive RM's SCI token returned to the caller. This is the SCI token for the target RM address space to which SCI sent the request.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. RM reason codes are defined in CSLRRR. SCI reason codes are defined in CSLSRR. Possible reason codes are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

CSLRMQRY return and reason codes

The following table lists the return and reason codes that can be returned on a CSLRMQRY request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 53. CSLRMQRY return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'03000004'	X'00001000'	No resources were found.
X'03000008'	X'00002000'	The client is not registered.
	X'00002100'	The query-list length is invalid.
	X'00002108'	The query-list address is invalid.
	X'00002110'	The list version in the list header (QRYL_PVER) is zero, which is invalid. The list version must be set in the list header to the maximum list version (QRYL_PVERMAX).
	X'00002114'	The list header length cannot be zero or greater than the list length that was passed in. The list header length (QRYL_HDRLEN) must be set in the list header to the list header length.
	X'00002404'	No resource structure is defined.
X'0300000C'	X'00003000'	The request succeeded for at least one list entry, but not all. Check the completion code in each query list entry in the OUTPUT buffer for individual errors.
	X'00003004'	The request failed for all entries. Check the completion code in each query list entry in the OUTPUT buffer for individual errors.
X'03000010'	X'00004000'	The CQS address space is unavailable. Retry the request again to attempt routing the request to another RM with an available CQS.
	X'00004100'	The requested version is not supported. The client compiled with a version of CSLRMQRY that is not supported by RM. All RMs must be migrated to a new release before IMS is migrated to a new release that uses a new CSLRMQRY function.
	X'00004104'	The list version is not supported. The client created the query list at a version that is not supported by RM. All RMs must be migrated to a new release before the client is migrated to a new release that uses a new CSLRMQRY function.
X'03000014'	X'00005000'	Storage allocation for the query output buffer failed.
	X'00005108'	Storage allocation for the CQSBRWSE buffer failed.
	X'00005200'	The CQS request resulted in an unexpected error.
	X'00005204'	The CQS request failed because RM incorrectly built the request input.

Related concepts

"Issue CSL RM requests to manage global resources" on page 164

Before a client can access or change global resource information, it must register to SCI using the CSLSCREG request. After the client registers to SCI, it must register to RM using the CSLRMREG request. The client must issue an SCI registration request for every IMSplex with which it intends to communicate.

CSLRMREG: register clients

You use the CSLRMREG request to register a client to RM and, optionally, to register the client's resource types and associated name types. The client being registered to RM must be authorized to issue a CSLRMREG request. However, you cannot register a client if an IMSplex-wide process is in progress.

You must register a client to RM before the client can issue any other RM requests. After the client is registered, it must participate in any IMSplex-wide processes that are performed. You must register the client to all RMs that are active in the IMSplex. If registration to an RM fails, you must deregister the client from any RMs to which the client had successfully registered. If an RM fails, register with it when it restarts.

You can register the same client multiple times. For example, you might need to specify the resource list for the client after the client is already registered. Optionally, register resource types to RM along with the client to define the resource types to RM and associate a name type with each resource type. You must register resource types before you can specify them in other requests. You cannot register the client if the resource type and name type associations do not match those already registered previously.

Resource-processing clients, such as the IMS control region, issue this request.

This request is supported in assembler language.

CSLRMREG syntax

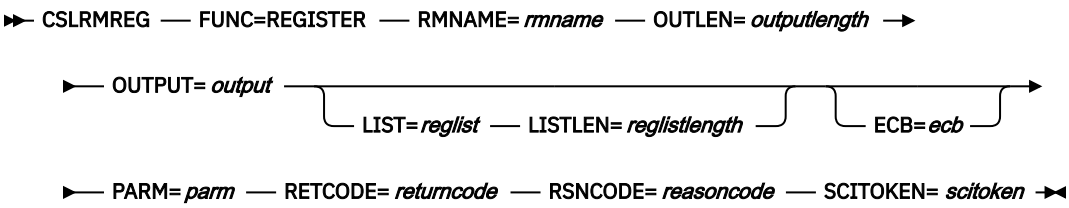
CSLRMREG DSECT syntax

Use the DSECT function of a CSLRMREG request to include the following inputs and outputs in your program:

- Equate (EQU) statements for the length of the CSLRMREG parameter list
- The CSLRMREG return codes, reason codes, and completion codes
- The CSLRREGL DSECT to map the input registration list
- The CSLRREGO DSECT to map the register output

CSLRMREG REGISTER syntax

Use the CSLRMREG request to register a client to RM and, optionally, to register the client's resource types and associated name types to RM.



CSLRMREG parameters

ECB=*symbol*
ECB=*(r2-r12)*
(Optional) - Specifies the address of a z/OS ECB used for asynchronous requests. When the request is complete, the ECB specified is posted. If an ECB is not specified, the task is suspended until the request is complete. If an ECB is specified, the invoker of the request must issue a WAIT (or

equivalent) after receiving control from CSLRMREG before using or examining any data returned by this request (including the RETCODE and RSNCODE fields).

LIST=*symbol*
LIST=*(r2-r12)*
(Optional) - Specifies the registration list built by the caller. Each list entry is a separate resource type registration. If a registration list is specified when no resource structure is defined, it is ignored.

The CSLRREGL macro maps the registration list entry. The list contains a header and one or more list entries. The list entries must reside in contiguous storage. Each registration list entry contains the following:

- Resource type
- Name type

LISTLEN=*symbol*
LISTLEN=*(r2-r12)*
(Optional) - Specifies the 4-byte registration list length. LISTLEN is required if LIST is specified.

OUTLEN=*symbol*
OUTLEN=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the length of the output returned by the CSLRMREG request. OUTLEN contains the length of the output pointed to by the OUTPUT= parameter.

The output length is zero if no output is built, for example, if an error is detected before any output can be built.

OUTPUT=*output*
OUTPUT=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the address of the variable length output returned by the CSLRMREG request. The output contains a header and zero, one, or more registration entries for registrations that were attempted. The output length is returned in the OUTLEN= field.

The output address is zero if no output was built, for example, if an error was detected before any output could be built.

The CSLRREGO macro maps the output that is returned. The output contains a header and zero, one, or more list entries. The output header contains the following:

- Eyecatcher
- Output length
- CSLRREGO version
- CSLRREGO header length (offset to start of entries)
- CSLRREGO entry length
- Registration list count
- Time stamp
- Registration status
- Structure version

Each output entry represents a registration request that was attempted. Each entry contains the following:

- Resource type
- Name type
- Completion code for the registration request. Completion codes are mapped by CSLRRR. Possible completion codes are:

X'00000000'
Register succeeded.

X'00000008'
Invalid resource type. The resource type cannot be zero.

X'0000000C'
Invalid name type. The name type cannot be zero, or the resource type is already defined with a different name type.

PARM=*symbol*
PARM=*(r2-r12)*
(Required) - Specifies the CSLRMREG parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by RREG_PARMLN.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - specifies a 4-byte field to receive the return code on output. RM return codes are defined in CSLRRR. SCI return codes are defined in CSLSRR. Possible return codes are described in the following table.

RMNAME=*symbol*
RMNAME=*(r2-r12)*
(Required) - Specifies an 8-byte RM name to which to send the registration request.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. RM reason codes are defined in CSLRRR. SCI reason codes are defined in CSLSRR. Possible reason codes are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token was returned by a successful `CSLSCREG FUNC=REGISTER` request.

CSLRMREG return and reason codes

The following table lists the return and reason codes that can be returned on a CSLRMREG request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 54. CSLRMREG return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'03000004'	X'00001100'	The request completed successfully but the LIST is ignored. No resource structure is defined.
X'03000008'	X'00002100'	The registration-list length is invalid.
	X'00002108'	The registration-list address is invalid.
	X'00002110'	The list version in the list header (REGL_PVER) is zero, which is invalid. The list version must be set in the list header to the maximum list version (REGL_PVERMAX).
	X'00002114'	The list header length cannot be zero or greater than the list length that was passed in. The list header length (REGL_HDRLEN) must be set in the list header to the list header length.
X'0300000C'	X'00003000'	The request is valid for at least one list entry, but not all. The registration for the valid list entries is not performed and the client registration is rejected. Check the completion code in each list entry in the OUTPUT buffer for individual errors.

Table 54. CSLRMREG return and reason codes (continued)

Return code	Reason code	Meaning
	X'00003004'	The request failed for all entries. Check the completion code in each list entry in the OUTPUT buffer for individual errors.
X'03000010'	X'00004010'	The client is not authorized.
	X'00004100'	The requested version is not supported. The client compiled with a version of CSLRMREG that is not supported by RM. All RMs must be migrated to a new release before IMS is migrated to a new release that uses a new CSLRMREG function.
	X'00004104'	The list version is not supported. The client created the registration list at a version that is not supported by RM. All RMs must be migrated to a new release before the client is migrated to a new release that uses a new CSLRMREG function.
X'03000014'	X'00005000'	Storage allocation for the register output buffer failed.
	X'00005100'	Storage allocation for CQSUPD buffer failed.
	X'00005200'	CQS request resulted in an unexpected error.
	X'00005204'	CQS request failed because RM incorrectly built request input.
	X'00005110'	The client block allocation failed.

Related concepts

"Sequence of RM client requests" on page 163
If you want to use RM to manage global resources in an IMSplex for your own product or service, you have to write one or more RM clients. An RM client uses RM requests issued in a particular sequence to communicate with RM.

"Issue CSL RM requests to manage global resources" on page 164
Before a client can access or change global resource information, it must register to SCI using the CSLSCREG request. After the client registers to SCI, it must register to RM using the CSLRMREG request. The client must issue an SCI registration request for every IMSplex with which it intends to communicate.

CSLRMUPD: update resources

By issuing the CSLRMUPD request, you can create a resource if it does not exist, or update a resource if it does exist (as long as the version specified matches the version of the resource). A resource can be created or updated with or without client data.

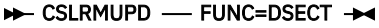
This request is supported in assembler language.

CSLRMUPD syntax

CSLRMUPD DSECT syntax

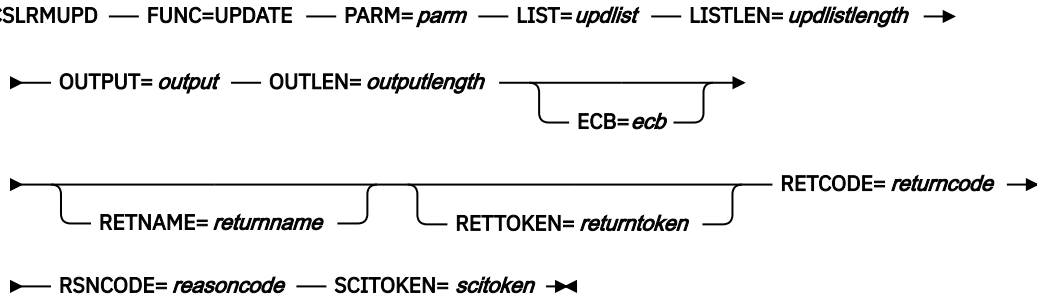
Use the DSECT function of a CSLRMUPD request to include the following inputs and outputs in your program:

- Equate (EQU) statements for the length of the CSLRMUPD parameter list
- The CSLRMUPD return codes, reason codes, and completion codes
- The CSLRUPDL DSECT to map the input update list
- The CSLRUPDO DSECT to map the update output



CSLRMUPD UPDATE syntax

Use the CSLRMUPD request to create or update a uniquely named resource on a resource structure.



CSLRMUPD parameters

ECB=*symbol*
ECB=*(r2-r12)*
(Optional) - Specifies the address of a z/OS ECB used for asynchronous requests. When the request is complete, the ECB specified is posted. If an ECB is not specified, the task is suspended until the request is complete. If an ECB is specified, the invoker of the request must issue a WAIT (or equivalent) after receiving control from CSLRM UPD before using or examining any data returned by this request (including the RETCODE and RSNCODE fields).

LIST=*symbol*
LIST=*(r2-r12)*
(Required) - Specifies the update resource list built by the caller. Each list entry is a separate update request. The list length can vary, depending upon the number of list entries and whether they contain DATA2.

The CSLRUPDL macro maps the update resource list entry. The list contains a header and one or more list entries. The list entries must reside in contiguous storage. Each update list entry contains the following:

- Entry length - the update list entry length. The list entry length can vary, depending upon whether DATA2 is specified.
- Resource name - client-defined name of the resource.
- Resource type - the resource type is a client-defined physical grouping of resources on the resource structure. Valid values are 1-255.
- Update options - options that indicate special processing to perform for the update.
- Version - the resource version, which is the number of times the resource has been updated. The version must match the resource's version for an existing resource for the update to succeed. The version must be zero to create a resource.
- Owner - owner of the resource.
- DATA1 - a small piece of client data (fixed length, contained in the adjunct area of a data entry) for the resource to be updated.
- DATA2 length - DATA2 length, if DATA2 is specified.
- DATA2 - a large piece of client data (variable length, contained in one or more data elements of a data entry) associated with the resource to be updated. DATA2 is optional. The maximum size of DATA2 is 61312 bytes (X'EF80').

LISTLEN=*symbol*
LISTLEN=*(r2-r12)*
(Optional) - Specifies the 4-byte update resource list length. LISTLEN is required if LIST is specified.

OUTLEN=*symbol*
OUTLEN=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the length of the output returned by the CSLRMUPD request. OUTLEN contains the length of the output pointed to by the OUTPUT= parameter.

The output length is zero if no output is built, for example, if an error is detected before any output can be built.

OUTPUT=*output*
OUTPUT=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the address of the variable length output returned by the CSLRMUPD request. The output contains a header and one or more update entries for resource updates that were attempted. The output length is returned in the OUTLEN= field.

The output address is zero if no output was built, for example, if an error was detected before any output could be built.

The CSLRUPDO macro maps the output that is returned. The output contains a header and one or more list entries. The output header contains the following:

- Eyecatcher
- Output length
- CSLRUPDO version
- Time stamp
- Resource entry count
- CSLRUPDO header length (offset to start of entries)
- Minimum entry length (offset to DATA2)

Each output entry represents a resource update that was attempted. Each entry contains the following:

- Output entry length - the list entry length can vary, depending upon whether DATA2 is returned.
- Resource type
- Name type - the name type is a client-defined value associated with a resource type that ensures uniqueness of client-defined resource names within a name type. Valid values are 1-255.
- Resource name
- Version - new resource version, if update succeeded, or the resource version of an existing resource, if the failed because of a version mismatch.
- Owner - resource owner of an existing resource, if the update failed because of a version mismatch and the option to read the owner was set.
- DATA1 - a small piece of client data (fixed length, contained in the adjunct area of a data entry) associated with an existing resource, if the update failed because of a version mismatch and the option to read DATA1 was set.
- DATA2 length - length of large piece of client data associated with an existing resource, if the update failed because of a version mismatch, DATA2 exists, and the option to read DATA2 was set.
- DATA2 - a large piece of client data (variable length, contained in one or more data elements of a data entry) associated with an existing resource, if the update failed because of a version mismatch, DATA2 exists, and the option to read DATA2 was set. The maximum size of DATA2 is 61312 bytes (X'EF80').
- Completion code for the update request - completion codes are mapped by CSLRRR. Possible completion codes are:

X'00000000'
Update request succeeded.

X'00000008'
Invalid resource type.

X'00000010'
Version mismatch. Resource already exists and version specified on input did not match.

X'00000014'
Resource already exists as a different resource type.

X'00000018'
Resource type is not registered. The resource type must be registered using a CSLRMREG request.

X'0000001C'
Resource structure is full.

X'00000024'
Resource structure is unavailable.

X'00000038'
Update failed because of CQS internal error.

X'0000003C'
Update failed because RM incorrectly built the CQSUPD list entry.

X'00000040'
Version mismatch. The resource already exists and the version specified on input did not match. The requestor requested that DATA2 be passed back, but RM encountered an error reading DATA2.

The output buffer is not preallocated by the caller. After the request returns it, this word contains the address of a buffer containing the update output. It is the caller's responsibility to release this storage by issuing the CSLSCBFR FUNC=RELEASE request when it is through with the storage. The length of the output is returned in the OUTLEN= field.

PARM=*symbol*
PARM=*(r2-r12)*
(Required) - Specifies the CSLRMUPD parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by RUPD_PARMLN.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - specifies a 4-byte field to receive the return code on output. RM return codes are defined in CSLRRR. SCI return codes are defined in CSLSRR. Possible return codes are described in the following table.

RETNAME=*symbol*
RETNAME=*(r2-r12)*
(Optional) - Specifies an 8-byte field to receive the RM name returned to the caller. This is the CSL member name of the target RM address space to which SCI sent the request.

RETTOKEN=*symbol*
RETTOKEN=*(r2-r12)*
(Optional) - Specifies a 16-byte field to receive RM's SCI token returned to the caller. This is the SCI token for the target RM address space to which SCI sent the request.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. RM reason codes are defined in CSLRRR. SCI reason codes are defined in CSLSRR. Possible reason codes are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

CSLRMUPD return and reason codes

The following table lists the return and reason codes that can be returned on a CSLRMUPD request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 55. CSLRMUPD return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'03000008'	X'00002000'	The client is not registered.
	X'00002100'	The update-list length is invalid.
	X'00002108'	The update-list address is invalid.
	X'0000210C'	One of the list entries contains one of the following invalid list entry lengths: • Zero length • Smaller than the minimum list entry length • Beyond the end of the list passed in • Not on a fullword boundary RM assumes that the rest of the list is invalid.
	X'00002110'	The list version in the list header (UPDL_PVER) is zero, which is invalid. The list version must be set in the list header to the maximum list version (UPDL_PVERMAX).
	X'00002114'	The list header length cannot be zero or greater than the list length that was passed in. The list header length (UPDL_HDRLEN) must be set in the list header to be the list header length.
	X'00002200'	One of the list entries contains an invalid resource type, such as zero. RM assumes the rest of the list is invalid.
	X'0000220C'	One of the entries in the list contains one or more invalid update options. RM assumes the rest of the list is invalid.
	X'00002404'	No resource structure is defined.
X'0300000C'	X'00003000'	The request succeeded for at least one list entry, but not all. Check the completion code in each list entry in the OUTPUT buffer for individual errors.
	X'00003004'	The request failed for all entries. Check the completion code in each list entry in the OUTPUT buffer for individual errors.
	X'00003008'	The request failed for one or more list entries and all failures were version mismatches. Check the completion code in each list entry in the OUTPUT buffer for individual errors.
X'03000010'	X'00004000'	The CQS address space is unavailable. Retry the request to attempt routing the request to another RM with an available CQS.
	X'00004100'	The requested version is not supported. The client compiled with a version of CSLRMUPD that is not supported by RM. All RMs must be migrated to a new release before IMS is migrated to a new release that uses a new CSLRMUPD function.
	X'00004104'	The list version is not supported. The client created the update list at a version level that is not supported by RM. All RMs must be migrated to a new release before the client is migrated to a new release that uses a new CSLRMUPD function.

Table 55. CSLRMUPD return and reason codes (continued)

Return code	Reason code	Meaning
X'03000014'	X'00005000'	Storage allocation for the output buffer failed. The resource updates might or might not have succeeded.
	X'00005100'	Storage allocation for CQSUPD buffer failed.
	X'00005200'	CQS request resulted in unexpected error.
	X'00005204'	The CQS request failed because RM incorrectly built the request input.

Related concepts

"Issue CSL RM requests to manage global resources" on page 164
Before a client can access or change global resource information, it must register to SCI using the CSLSCREG request. After the client registers to SCI, it must register to RM using the CSLRMREG request. The client must issue an SCI registration request for every IMSplex with which it intends to communicate.

CSL RM directives

An RM directive is a function that RM defines that can be sent as a message to RM clients, informing the RM clients of work to be processed. After a resource processing client is registered to RM, RM can direct that client to perform RM functions, or directives, by issuing the CSLSCMSG request. A resource processing client is any system that manages resources and uses RM to manage global information about those resources.

RM directives are always issued in message protocol (PROTOCOL=MSG), that is, asynchronously; RM therefore expects no response from the RM client, and it continues processing without waiting for a response. The RM client is responsible for determining whether or not to take any action in response to the directive. If the client does not respond, the directive times out.

The CSLRMDIR macro maps the RM directives. The SCI Input exit routine's INXP_MBRPLPTR field points to the CSLRMDIR parameter list.

The following RM directives are defined in the CSLRMDIR macro:

- Repopulate structure (RDIR_STRPOPD)
- Structure failed (RDIR_STRFAILD)
- Process step (RDIR_PRSTEPD)
- Process step response (RDIR_PRRESPD)

Related reference

CSL SCI Input exit routine (Exit Routines)

CSL RM repopulate structure directive

The repopulate structure directive is sent to all resource processing clients after an RM detects a structure failure and the structure is reallocated. The client then repopulates the structure and can receive this directive from all RMs that it is registered to.

If the client receives directives from multiple RMs to repopulate the structure after having already done so, it can ignore those requests after confirming that the directives apply to the same structure name and version.

Repopulate structure parameters

RDIR_STRPOP
Identifies the start of the repopulate structure directive.

RDIR_STNAMLEN=*length*
Contains the length of the structure name.

RDIR_STNAMPTR=*address*
Contains the address of the structure name.

RDIR_STVERLEN=*length*
Contains the length of the structure version.

RDIR_STVERPTR=*address*
Contains the address of the structure version.

RDIR_STRPOPLN=*length*
Contains the length of the repopulate structure.

CSL RM structure failed directive

The structure failed directive is sent to a resource processing client when the resource structure fails and cannot be reallocated. The client cannot make any more resource requests until the problem is corrected.

A client can receive this directive from all RMs to which it is registered. If the client receives directives from multiple RMs, it can ignore duplicate requests after confirming that the directives apply to the same structure name and version.

Structure failed parameters

RDIR_STRFAIL
Identifies the start of the structure failed directive.

RDIR_SFNAMLEN=*length*
Contains the length of the structure name.

RDIR_SFNAMPTR=*address*
Contains the address of the structure name.

RDIR_SFVERLEN=*length*
Contains the length of the structure version.

RDIR_SFVERPTR=*address*
Contains the address of the structure version.

RDIR_STRFAILN=*length*
Contains the length of the structure failed directive.

CSL RM process step directive

The Process Step directive is sent to a resource processing client when a process step needs to be performed.

Process step parameters

RDIR_PRSTEP
Identifies the start of the Process Step directive.

RDIR_PSTKNLEN=*length*
Contains the length of the process token (PRCTOKEN), which uniquely identifies the IMSplex-wide process. PRCTOKEN is returned after the CSLRMPRI FUNC=INITIATE request successfully completes. PRCTOKEN can be specified on CSLRMPRS FUNC=PROCESS, CSLRMPRR FUNC=RESPOND, and CSLRMPRT FUNC=TERMINATE requests.

RDIR_PSTKNPTR=*address*
Contains the address of the PRCTOKEN.

RDIR_PSUOWLEN=*length*
Contains the length of the UOWTOKEN, a client-defined UOW that uniquely identifies a process instance. UOWTOKEN also unites the PROCESS INITIATE, PROCESS RESPOND, and PROCESS

TERMINATE steps. UOWTOKEN is defined by the CSLRMPRI FUNC=INITIATE request and can be specified on CSLRMPRS FUNC=PROCESS requests.

RDIR_PSUOWPTR=*address*
Contains the address of the UOWTOKEN.

RDIR_PRCNMLEN=*length*
Contains the length of the process name (PRCNAME), which is defined by the CSLRMPRI FUNC=INITIATE request. It can also be specified on the CSLRMPRS FUNC=PROCESS and CSLRMPRT FUNC=TERMINATE requests.

RDIR_PRCNMPTR=*address*
Contains the address of the PRCNAME.

RDIR_PRCTYPE
The process type is defined by the CSLRMPRI FUNC=INITIATE request. It can be specified on the CSLRMPRS FUNC=PROCESS and CSLRMPRT FUNC=TERMINATE requests. This parameter is passed by value; the length field is always zero.

RDIR_PSNAME
Contains the process step name, which is defined by the CSLRMPRS FUNC=PROCESS request. This parameter is passed by value; the length field is always zero.

RDIR_PSDATLEN=*length*
Contains the length of the process step client data (CDATALEN). The client data is passed to the participants in the process step. CDATALEN is specified on the CSLRMPRS FUNC=PROCESS request.

RDIR_PSDATPTR=*address*
Contains the address of the process step client data (CDATA).

RDIR_CNAMLEN=*length*
Contains the length of the client name that was registered to SCI by the client that originated the process step (the process step master).

RDIR_CNAMPTR=*address*
Contains the address of the client name that was registered to SCI by the client that originated the process step (the process step master).

RDIR_CTYPE
Identifies the client type that was registered to SCI by the client that originated the process step (the process step master). This parameter is passed by value; the length field is always zero.

RDIR_CSTYPLEN
Contains the length of the client subtype that was registered to SCI by the client that originated the process step (the process step master).

RDIR_CSTYPPTR
Contains the address of the client subtype that was registered to SCI by the client that originated the process step (the process step master).

RDIR_PRSTEPLN
Contains the length of the process step directive.

CSL RM process step response directive

The Process Step Response directive is sent to RM by a client that is responding to a process step with a CSLRMPRR request.

Process step response parameters

RDIR_PRRESP
Identifies the start of the process step response directive.

RDIR_PRTKNLEN=*length*
Contains the length of the process token (PRCTOKEN), which uniquely identifies the IMSplex-wide process. PRCTOKEN is returned after the CSLRMPRI FUNC=INITIATE request successfully completes.

PRCTOKEN can be specified on CSLRMPRS FUNC=PROCESS, CSLRMPRR FUNC=RESPOND, and CSLRMPRT FUNC=TERMINATE requests.

RDIR_PRTKNPTR=*address*
Contains the address of the PRCTOKEN.

RDIR_PROUTLEN=*length*
Contains the length of the process step response output (OUTPUT). The response output is passed back to the originator of the process step. OUTPUT is specified on the CSLRMPRR FUNC=RESPOND request.

RDIR_PROUTPTR=*address*
Contains the address of the response output (OUTPUT).

RDIR_PRRCLEN=*length*
Contains the process step response return code (RQSTRC). The return code is specified by the CSLRMPRR FUNC=RESPOND request.

RDIR_PRRCPTR=*address*
Contains the address of the process step response return code (RQSTRC).

RDIR_PRRSNLEN=*length*
Contains the length of the process step response reason code (RQSTRSN), which is specified by the CSLRMPRR FUNC=RESPOND request.

RDIR_PRRSNPTR=*address*
Contains the address of the process step response reason code (RQSTRSN).

Chapter 8. Writing a CSL SCI client

You must establish a connection to SCI (Structured Call Interface) in order to write a program that participates in an IMSplex (such as an AOP) and allows your IMSplex member to communicate with other IMSplex members. Without a connection to SCI, a program cannot participate in an IMSplex and communicate with other IMSplex members.

To establish a connection with SCI, you can use a subset of the SCI requests. These requests establish or terminate a connection with SCI and optionally indicate to SCI that the IMSplex member is in a ready state. When a member is in a ready state, it can have requests and messages routed to it by type.

SCI requests are also used by an IMSplex member to communicate with other IMSplex members and to find out information about those members. IMSplex members communicate with other members by using SCI requests to send messages, requests, and responses to requests. A query request can be used to find out information about the other members of the IMSplex.

Related reference

"CSLSCQRY: query request" on page 210
By issuing the CSLSCQRY request, an IMSplex member can obtain information about the members of the IMSplex.

Sequence of CSL SCI requests

Structured Call Interface (SCI) requests must be issued in a particular sequence in order to successfully register to SCI, ready the member, release storage allocated for the member, quiesce the member, and deregister the member from SCI.

The first SCI request is CSLSCREG. The member can then issue CSLSCRDY to tell SCI that it is ready to receive messages and requests that are routed by member type. If a member has storage that is allocated by SCI (for example, a message or an SCI allocated output parameter is received), the SCI buffer release request, CSLSCBFR, is issued to release the storage.

When a member is ready to terminate, the SCI quiesce request, CSLSCQSC, is used to tell SCI that the member does not want to receive messages and requests that are routed by member type. After the SCI deregistration request, CSLSCDRG, is used to terminate the connection with SCI, the member can no longer participate in the IMSplex.

The following table lists the sequence of requests issued by an SCI client. The request is listed with its purpose.

Table 56. Sequence of requests for SCI client

Request	Purpose
CSLSCREG	Register to SCI, which establishes the connection with SCI and enables the member to communicate within the IMSplex.
CSLSCRDY	Readies the member to SCI, which allows SCI to route messages and requests that are routed by member type to this member.
CSLSCBFR	Releases storage allocated for the member by SCI (for example, message data or parameters allocated by SCI from a request).
CSLSCQSC	Quiesces the member to SCI, which tells SCI not to route messages and requests that are routed by member type to this member.
CSLSCDRG	Deregisters the member from SCI which ends the member's connection with SCI.

Advanced CSL SCI requests

After connecting to Structured Call Interface (SCI), an IMSplex member can use advanced SCI requests to request services from and find out information about other IMSplex members. Each advanced request has its own purpose.

After establishing the connection with SCI, an IMSplex member can use advanced SCI requests to:

- Communicate, or request services, from other IMSplex members.

  A message protocol and a request protocol are provided to facilitate communication among IMSplex members. A message is a one-way communication with another IMSplex member. A request requires that a response be returned to the requesting member.
- Find out information about the other members in the IMSplex.

  A query request, CSLSCQRY, allows an IMSplex member to find out who the other members of the IMSplex are and to obtain information about those IMSplex members.

The following table lists the advanced SCI requests with their purpose. These requests can be issued without regard to sequence; however, the IMSplex member issuing the request must have registered to SCI.

Table 57. Advanced SCI requests for IMSplex members

Request	Purpose
CSLSCMSG	Sends a one-way message to another IMSplex member.
CSLSCRQS	Sends a request to another IMSplex member. SCI expects a response to the request.
CSLSCRQR	Sends a response to a previously issued request.
CSLSCQRY	Issues a query to SCI to find out information about members of the IMSplex.

CSL SCI requests

SCI requests can be issued by an IMSplex member. Any member can also receive messages from any other IMSplex member after a connection is established.

Related concepts

"CSL OM automated operator program clients" on page 125

OM provides an API interface for application programs that automate operator actions known as automated operator programs (AOP). You can use an AOP to issue commands that are embedded in an OM API request to an OM.

CSLSCBFR: buffer return request

The CSLSCBFR request releases the storage that the Structured Call Interface (SCI) allocated for an IMSplex member. This storage is allocated to receive either an input message sent from another IMSplex member with the CSLSCMSG request, or an output parameter generated from a CSLSCRQS request.

Another macro can invoke the CSLSCRQS request as part of the code generated by the macro which, in turn, can return an SCI data type. The storage allocated for these parameters must be released with the CSLSCBFR macro. The CSLSCQRY macro is an example of an SCI macro that does this. The OUTPUT parameter specifies the address in storage to receive the address of the buffer that contains the output from the CSLSCQRY macro. Release this storage by using the CSLSCBFR request.

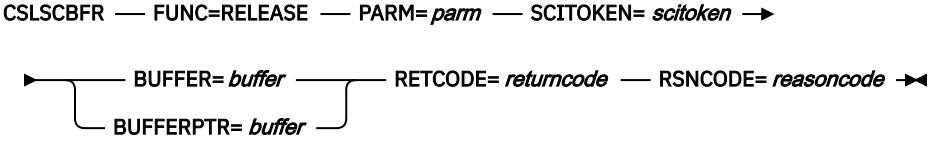
Syntax

DSECT syntax

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

RELEASE syntax

Use the RELEASE function of the CSLSCBFR request to release an SCI message buffer or SCI data type buffer. The SCI data type buffer is used for selected output parameters of the CSLSCRQS request for which SCI allocates storage.



For messages generated from a CSLSCMSG request, the buffer address is the address of the member parameter list that is specified to the member input exit in the INXP_MBRPLPTR field in the input exit parameter list.

For a response generated from a CSLSCRQS request that uses an SCI data type buffer, the storage is allocated when the request is returned to the IMSplex member that initiated the original request. The buffer address is the address of this storage, which is returned in the field specified by the member on the request.

After the CSLSCBFR request is complete, the storage contained in the message buffer or request response is no longer accessible by the IMSplex member.

For non-authorized members, the storage must be released from a TCB that is under the JOBSTEP TCB from which the SCI registration call was made. The release fails if it is done from a TCB that is not under the registered JOBSTEP TCB.

Parameters

BUFFER=*symbol*
BUFFER=*(r1-r12)*
4-byte parameter that contains the address of a buffer that is to be released.

Either BUFFER or BUFFERPTR is required.

BUFFERPTR=*symbol*
BUFFERPTR=*(r1-r12)*
4-byte parameter that contains the address of a word in storage that contains the address of the buffer that is to be released.

Either BUFFER or BUFFERPTR is required.

PARM=*symbol*
PARM=*(r1-r12)*
Specifies the CSLSCBFR parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by SBFR_PARMLN.

RETCODE=*symbol*
RETCODE=*(r1-r12)*
Specifies a 4-byte field to receive the return code on output. The SCI return codes are defined in CSLSRR. Possible return codes for CSLSCBFR are described in the following table.

RSNCODE=*symbol*
RSNCODE=*(r1-r12)*
Specifies a 4-byte field to receive the reason code on output. The SCI reason codes are defined in CSLSRR. Possible reason codes for CSLSCBFR are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r1-r12)*
Specifies a 16-byte field containing the SCITOKEN. This token uniquely identifies this IMSplex member's connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

Return and reason codes

The following table lists the return and reason codes that can be returned on a CSLSCBFR macro request.

Table 58. CSLSCBFR request return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'01000008'	X'00002014'	The buffer being released is not an SCI buffer.
	X'00002018'	Invalid SCI token.
	X'00002038'	Parameter list version is invalid.
	X'00002054'	The buffer being released is not an allocated buffer.
X'01000010'	X'00004FFF'	Function is not supported.
X'01000014'	X'00005000'	An SCI internal error occurred.
	X'00005074'	Buffer prefix is damaged on a CSLSCBFR call.
	X'00005078'	STORAGE RELEASE failed for SCI buffer on a CSLSCBFR call.
	X'00005500'	An abend occurred during CSLSCBFR processing.

CSLSCDRG: deregistration request

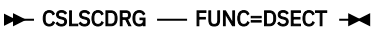
By issuing the CSLSCDRD request, you can terminate the connection between the IMSplex member and SCI. After successful completion of this request, the SCI token is no longer valid.

To make subsequent SCI requests, the IMSplex member must create a new connection with SCI with a CSLSCREG request.

CSLSCDRG syntax

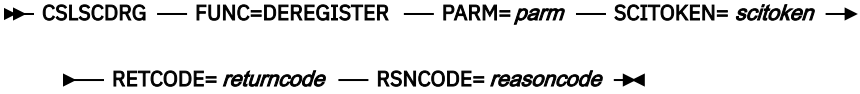
CSLSCDRG DSECT syntax

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



CSLSCDRG DEREGISTER syntax

The CSLSCDRG FUNC=DEREGISTER request deregisters the IMSplex member from SCI. After successful completion of the CSLSCDRG FUNC=DEREGISTER request, the SCITOKEN is invalid.



CSLSCDRG parameters

PARM=*symbol*
PARM=*(r1-r12)*
(Required) - Specifies the CSLSCDRG parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by SDRG_LN.

RETCODE=*symbol*
RETCODE=*(r1-r12)*
(Required) - Specifies the address of a 4-byte field to receive the CSLSCDRG return code. The SCI return codes are defined in CSLSRR. Possible return codes for CSLSCDRG are described in the following table.

RSNCODE=*symbol*
RSNCODE=*(r1-r12)*
(Required) - Specifies the address of a 4-byte field to receive the CSLSCDRG reason code. The SCI reason codes are defined in CSLSRR. Possible reason codes for CSLSCDRG are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r1-r12)*
(Required) - Specifies a 16-byte field containing the SCITOKEN. This token uniquely identifies this IMSplex member's connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

CSLSCDRG return and reason codes

The following table lists the return and reason codes that can be returned on a CSLSCDRG macro request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 59. CSLSCDRG return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'01000004'	X'00001010'	z/OS cross-system coupling facility leave for member failed.
X'01000008'	X'00002018'	Invalid SCI token.
	X'00002038'	Parameter list version is invalid.
X'01000010'	X'00004000'	SCI is not active.
	X'00004014'	CSLSDR00 could not be loaded.
	X'00004018'	There are still outstanding requests during deregistration.
	X'00004FFF'	Function is not supported.
X'01000014'	X'00005000'	An SCI internal error occurred.
	X'00005004'	SCI was unable to add the ESTAE routine.
	X'00005008'	A BPE SVC error occurred.
	X'00005020'	An ENQ resource error occurred.
	X'00005500'	An abend occurred during CSLSCDRG processing.

CSLSCMSG: send message request

By issuing the CSLSCMSG request, you can send a message to one or more other IMSplex members. The target members are specified by SCITOKEN, member name, or member type.

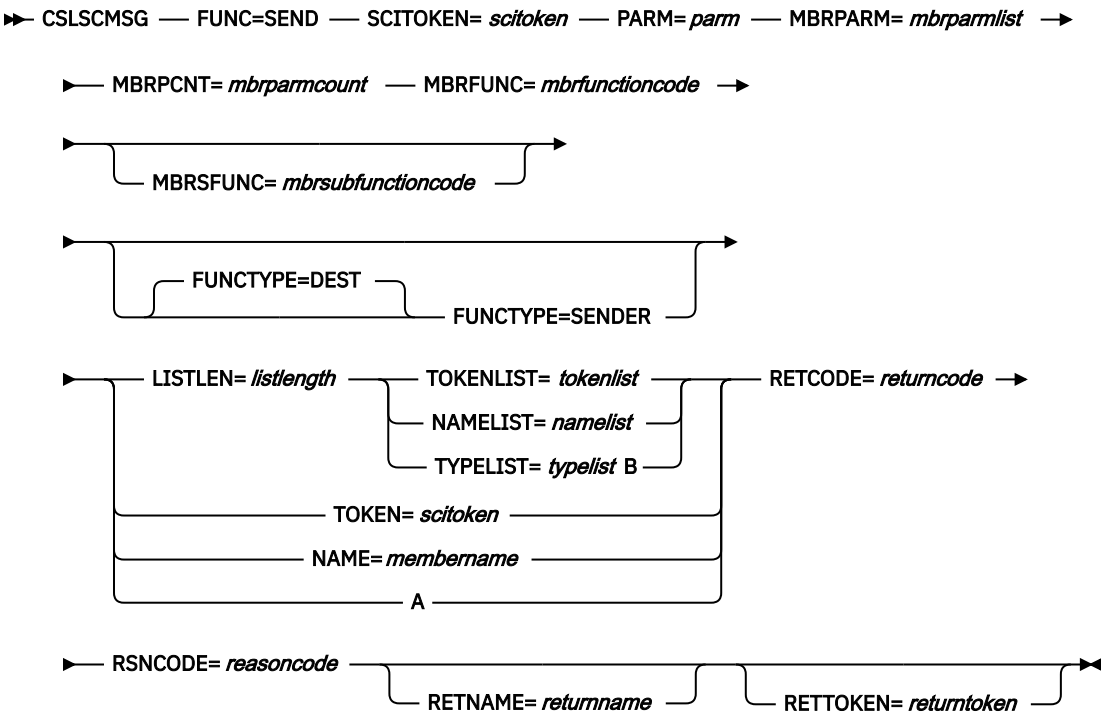
CSLSCMSG syntax

CSLSCMSG DSECT syntax

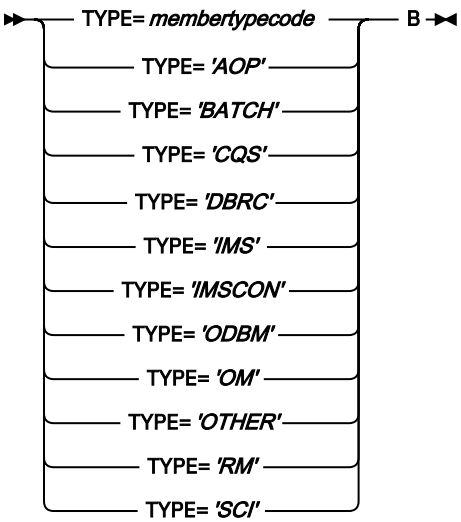
Use the DSECT function of a CSLSCMSG request to include equate (EQU) statements in your program for the CSLSCMSG parameter list length, the IMSplex types, and the CSLSCMSG return and reason codes.

CSLSCMSG SEND MESSAGE syntax

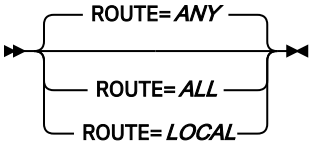
The syntax of the CSLSCMSG FUNC=SEND request is shown below:



A



B



CSLSCMSG parameters

FUNCTYPE=SENDER
FUNCTYPE=DEST
(Optional) - Specifies that the MBRFUNC and MBRSFUNC are defined by the DEST (destination) of this message or the SENDER of the message. This indicator is passed to the recipient of the message in the SCI Input exit parameter list.

LISTLEN=*<numeric literal>*
LISTLEN=*symbol*
LISTLEN=*(r1-r12)*
(Required if NAMELIST, TOKENLIST or TYPELIST is specified) - Specifies the length of the routing list. The routing list consists of a header and one or more list entries, each entry describing a single message destination (NAMELIST and TOKENLIST) or set of destinations (TYPELIST).

If LISTLEN is a numeric literal, all characters must be numbers. If any character is alphabetic, the parameter will be considered a symbol.

MBRFUNC=*symbol*
MBRFUNC=*(r1-r12)*
(Required) - Specifies a 4-byte member function code that is passed to the destination of the message in the SCI Input exit parameter list. This function code, along with the MBRSFUNC, identifies the message that is being sent.

If MBRFUNC is a symbol, the symbol points to a 4-byte area of storage that contains the function code.

MBRPARM=*symbol*
MBRPARM=*(r1-r12)*
(Required) - Specifies the address of a prebuilt parameter list. This parameter list must be built by the messaging module and consists of sets of pairs. Each pair describes a single parameter in the member parameter list and consists of the following:

parameter length
Four-byte parameter that specifies the length of the member parameter.

parameter address
Four-byte parameter that specifies the address of the member parameter.

The two methods for passing parameters in a parameter list are *by address* and *by value*. Both of these methods can be used when passing parameters in a CSLSCMSG request. The pair must be setup so that SCI will handle the parameter properly.

- By address

  To pass a parameter by address, the address of the parameter must be passed in *parameteraddress* and the length of the parameter must be passed in *parameterlength*. SCI will obtain the parameter from *parameteraddress*.

- By value

  To pass a parameter by value, the parameter must be passed in *parameteraddress* and zero must be passed in *parameterlength*. When the length is zero, SCI will copy the value contained in *parameteraddress* to the destination.

Member Parameter List: The user parameters specified here are presented to the IMSplex member that receives the message in the member parameter list, the address of which is contained in the

Input exit parameter area field INXP_MBRPLPTR. Each parameter is represented by eight bytes, the first four bytes contain *parameterlength* and the second four bytes contain *parameteraddress* (if *parameteraddress* is an address, the second four bytes point to storage in the local address space, not the requesting address space).

Null Parameters: In some cases, the message processing module expects a set number of parameters with a defined order. If a message is to be sent that does not contain all the parameters, null parameters must be sent to ensure the data buffer contains everything that is expected. Null parameters can be sent by specifying zero for *parameterlength* and *parameteraddress*. The eight bytes that represent the parameter in the data buffer will contain zeros.

MBRPCNT=*symbol*
MBRPCNT=*(r1-r12)*
(Required) - Specifies a 4-byte field that contains the number of member parameters that are included in MBRPARM.

MBRSFUNC=*symbol*
MBRSFUNC=*(r1-r12)*
(Optional) - Specifies a 4-byte member subfunction code that is passed to the destination of the message in the SCI Input exit parameter list. This subfunction code, along with the MBRFUNC, identifies the message that is being sent.

If MBRSFUNC is a symbol, the symbol points to a 4-byte area of storage that contains the subfunction code.

NAME=*symbol*
NAME=*(r1-r12)*
(Optional) - Specifies the address of an 8-byte member name of the destination of this message. This name can be obtained from the Notify exit (when the member joins the IMSplex) or by issuing a CSLSCQRY message.

Note: One of the routing parameters (NAME, TOKEN, TYPE, NAMELIST, TOKENLIST or TYPELIST) must be included.

To route by NAME, the destination member must be authorized. If the member is not authorized, the message is not sent.

NAMELIST=*symbol*
NAMELIST=*(r1-r12)*
(Optional) - Specifies the address of a list of member names to which this message is to be routed. This list consists of a header and one or more list entries, each entry defining a single member name. If NAMELIST is specified, LISTLEN must also be specified.

Note: One of the routing parameters (NAME, TOKEN, TYPE, NAMELIST, TOKENLIST or TYPELIST) must be included.

The list header DSECT is CSLSMGLH, and the list entry DSECT is CSLSNMLE. These DSECTs are defined in CSLSCMAP.

For a message to be routed to a member using NAMELIST, that member must be an authorized member. If a member name for a non-authorized member is included in NAMELIST, the name will not be found and the message will not be sent to that member.

The NAMELIST is sent to SCI for processing. Then, control is returned to your program. A response of "Request completed successfully" does not mean that the message was sent to all names in the list; it means that the list was successfully sent to SCI. Errors could occur while the list is processed and the message is sent. Possible errors include:

- Name not found
- Name found, but the member terminated before message is sent
- SCI abended

These errors are not returned to your program.

PARM=*symbol*
PARM=*(r1-r12)*
(Required) - Specifies the address of a parameter list used by the message to pass the parameters to SCI. The length of the storage must be at least equal to the value of SMSG_LN. The storage must begin on a word boundary.

RETCODE=*symbol*
RETCODE=*(r1-r12)*
(Required) - Specifies the address of a 4-byte field to receive the CSLSCMSG return code. The SCI return codes are defined in CSLSRR.

RETNAME=*symbol*
RETNAME=*(r1-r12)*
(Optional) - Specifies the address of an 8-byte field to receive the name of the IMSplex member to which the message was sent. If the message is sent to more than one destination, nothing is returned in this field.

RETTOKEN=*symbol*
RETTOKEN=*(r1-r12)*
(Optional) - Specifies the address of an 8-byte field to receive the token of the IMSplex member to which the message was sent. If the message is sent to more than one destination, nothing is returned in this field.

ROUTE=*ANY*
ROUTE=*ALL*
ROUTE=*LOCAL*
(Optional) - Specifies how the message should be routed to the type specified in the TYPE parameter or the types specified in the TYPELIST parameter. This parameter is valid only if TYPE or TYPELIST is specified.

ANY
Routes the message to a single member of the types specified. SCI selects the member that will receive the message. TYPE=ANY is not valid with TYPELIST.

ALL
Routes the message to all members of the specified types.

LOCAL
Routes the message to all members of the specified types that are active on the local z/OS image.

RSNCODE=*symbol*
RSNCODE=*(r1-r12)*
(Required) - Specifies the address of a 4-byte field to receive the CSLSCMSG reason code. The SCI reason codes are defined in CSLSRR.

SCITOKEN=*symbol*
SCITOKEN=*(r1-r12)*
(Required) - Specifies the address of a 16-byte field that contains the SCI token of the member making the request. The token was returned on the CSLSCREG request.

TOKEN=*symbol*
TOKEN=*(r1-r12)*
(Optional) - Specifies the address of the 16-byte SCI token of the destination of this message. This token can be obtained from the Notify exit (when the member joins the IMSplex) or by issuing a CSLSCQRY message.

Note: One of the routing parameters (NAME, TOKEN, TYPE, NAMELIST, TOKENLIST or TYPELIST) must be included.

TOKENLIST=*symbol*
TOKENLIST=*(r1-r12)*
(Optional) - Specifies the address of a list of SCI tokens that represent members to which this message is to be routed. This list consists of a header and one or more list entries, each entry defining a single SCI token. If TOKENLIST is specified, LISTLEN must also be specified.

Note: One of the routing parameters (NAME, TOKEN, TYPE, NAMELIST, TOKENLIST or TYPELIST) must be included.

The list header DSECT is CSLSMGLH, and the list entry DSECT is CSLSTKLE. These DSECTs are defined in CSLSCMAP.

The TOKENLIST is sent to SCI for processing. Then, control is returned to your program. A response of "Request completed successfully" does not mean that the message was sent to all SCI tokens in the list; it means that the list was successfully sent to SCI. Errors could occur while the list is processed and the message is sent. Possible errors include:

- Token not found
- Token found but member terminated before message is sent
- SCI abended

These errors are not returned to your program.

TYPE=*symbol*
TYPE=*'AOP'*
TYPE=*'BATCH'*
TYPE=*'CQS'*
TYPE=*'DBRC'*
TYPE=*'IMS'*
TYPE=*'IMSCON'*
TYPE=*'ODBM'*
TYPE=*'OM'*
TYPE=*'OTHER'*
TYPE=*'RM'*
TYPE=*'SCI'*

(Optional) - TYPE specifies the SCI type of the destination of this message. SCI routes the message to one or more members of the specified type (depending on the value of the route parameters). If there are no members of the specified type, an error is returned.

Note: One of the routing parameters (NAME, TOKEN, TYPE, NAMELIST, TOKENLIST or TYPELIST) must be included.

If this parameter is passed as a literal, the literal must be enclosed in single quotation marks. If this parameter is passed as a symbol or register, the symbol or register must contain the member type code. The member type code can be obtained by using the CSLSTPIX macro.

TYPELIST=*symbol*
TYPELIST=*(r1-r12)*

(Optional) - Specifies the address of a list of member types to which this message is to be routed. This list consists of a header and one or more list entries, each entry defining a single SCI token. If TYPELIST is specified, LISTLEN must also be specified.

Note: One of the routing parameters (NAME, TOKEN, TYPE, NAMELIST, TOKENLIST or TYPELIST) must be included.

The list header DSECT is CSLSMGLH, and the list entry DSECT is CSLSTPLE. These DSECTs are defined in CSLSCMAP.

The TYPELIST is sent to SCI for processing. Then, control is returned to your program. A response of "Request completed successfully" does not mean that the message was sent to all types in the list; it means that the list was successfully sent to SCI. Errors could occur while the list is processed and the message is sent. Possible errors include:

- No members of the specified type are active
- A member of the specified type was found but terminated before the message is sent
- SCI abended

These errors are not returned to your program.

CSLSCMSG return and reason codes

The following table lists the return and reason codes that can be returned on a CSLSCMSG macro request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 60. CSLSCMSG return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'01000008'	X'00002004'	An invalid function was passed to the SCI interface PC routine.
	X'00002008'	The number of parameters passed was either less than or equal to zero, or greater than the maximum allowed.
	X'00002010'	An invalid type was passed.
	X'00002018'	The SCI token was invalid.
	X'00002024'	The PHDR length was invalid.
	X'00002028'	The routing data length was invalid.
	X'00002034'	The length of the parameters is too large for a non-authorized caller.
	X'00002038'	The parameter list version is invalid.
X'01000010'	X'00004000'	SCI is not active.
	X'0000400C'	The destination IMSplex member is not active. The requested member might have been specified by name, token, or type.
	X'0000401C'	The calling member is in the process of deregistering from SCI.
	X'00004FFF'	The function is not supported.
X'01000014'	X'00005000'	An SCI internal error occurred.
	X'00005004'	An ESTAE add error occurred.
	X'00005024'	An error in the SRB routine occurred.
	X'00005028'	The routing type was invalid.
	X'0000502C'	The member could not be found due to an internal BPE hash table services error.
	X'00005030'	An SCI buffer could not be obtained.
	X'00005034'	A key 7 buffer in the SCI address space could not be obtained for a copy of PHDR and parameters.
	X'00005038'	An IEAMSCHD error occurred; the SRB could not be scheduled to the target address space.
	X'0000504C'	The message SRB key 7 parameter area could not be obtained.
	X'00005500'	An abend occurred during CSLSCMSG processing.
	X'00005504'	An abend occurred when the member parameters were copied to the target address space.

Related reference

"CSLSCMSG: send message request" on page 204

By issuing the CSLSCMSG request, you can send a message to one or more other IMSplex members. The target members are specified by SCITOKEN, member name, or member type.

"CSLSCRDY: ready request" on page 214
The SCI ready request enables the IMSplex member to receive messages and requests that are routed by TYPE. After the CSLSCREG request is issued and until CSLSCRDY is issued, the IMSplex member can only receive requests that are routed directly to a single target address space. The IMSplex member can send messages and requests that are routed by any method.

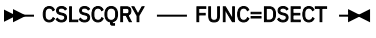
CSLSCQRY: query request

By issuing the CSLSCQRY request, an IMSplex member can obtain information about the members of the IMSplex.

CSLSCQRY syntax

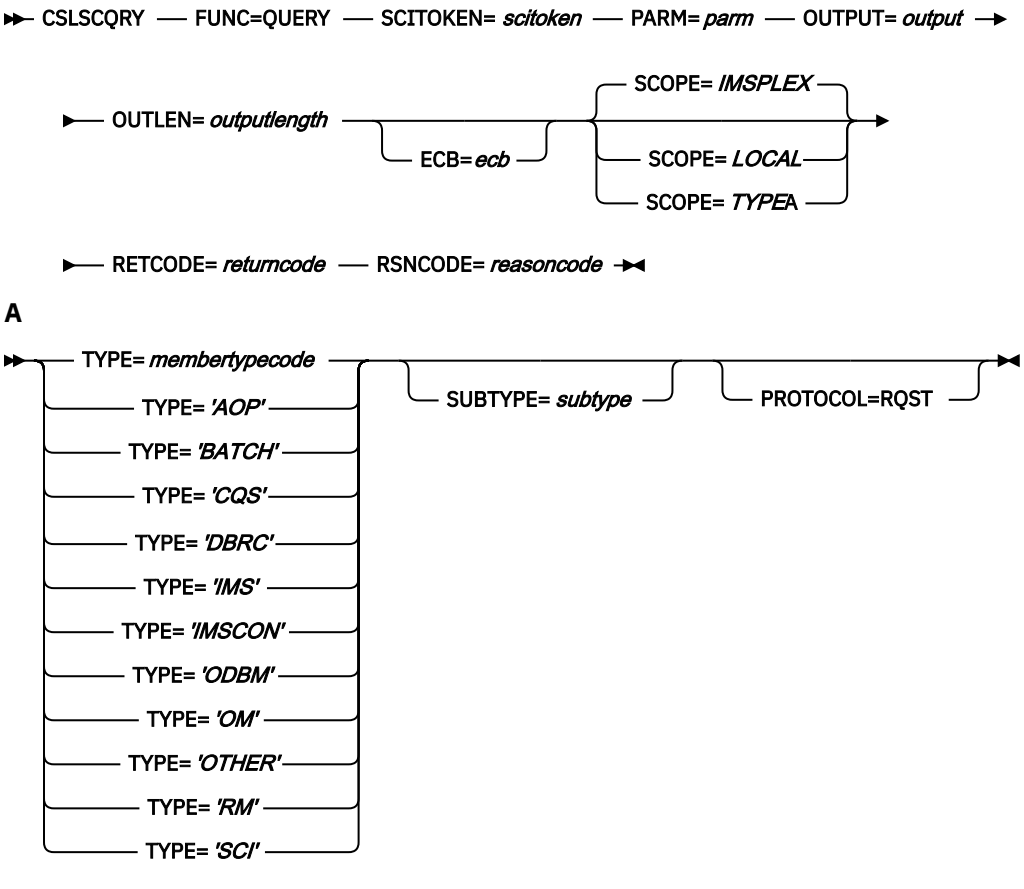
CSLSCQRY DSECT syntax

Use the DSECT function of a CSLSCQRY request to include equate (EQU) statements in your program for the CSLSCQRY parameter list length, the IMSplex types and the CSLSCQRY return and reason codes.



CSLSCQRY QUERY syntax

Use the following syntax to issue the CSLSCQRY service request. The output is returned to the caller when the request is complete.



CSLSCQRY parameters

ECB=*symbol*
ECB=*(r1-r12)*
(Optional) - Specifies the address of a z/OS ECB used for asynchronous requests. When the request is complete, the ECB specified is posted. If an ECB is not specified, the task is suspended until the request is complete. If an ECB is specified, the invoker of the macro must issue a WAIT (or equivalent) after receiving control from CSLSCQRY, before using or examining any data returned by this macro (including the RETCODE and RSNCODE fields).

OUTLEN=*symbol*
OUTLEN=*(r1-r12)*
(Required) - Specifies a 4-byte field to receive the length of the output returned by the CSLSCQRY request. OUTLEN receives the length of the output pointed to by the OUTPUT= parameter.

The output length is zero if no output is built, for example, if an error is detected before any output can be built.

OUTPUT=*output*
OUTPUT=*(r1-r12)*
(Required) - Specifies a field to receive a pointer to the variable length output returned by the CSLSCQRY request. The output length is returned in the OUTLEN= field.

The output address is zero if no output was built, for example, if an error was detected before any output could be built.

The CSLSQRYO macro maps the output that is returned. The output contains a header and one or more list entries.

The output buffer is not preallocated by the caller. After being returned by the request, this word contains the address of a buffer containing the query output. It is the caller's responsibility to release this storage by issuing the CSLSCBFR FUNC=RELEASE request when it is through with the storage.

PARM=*symbol*
PARM=*(r1-r12)*
(Required) - Specifies the CSLSCQRY parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by SQRY_PARMLN.

PROTOCOL=RQST
(Optional) - SCI protocol for sending the request to SCI. RQST indicates that the SCI request interface protocol is to be used for the request.

RETCODE=*symbol*
RETCODE=*(r1-r12)*
(Required) - Specifies the address of a 4-byte field to receive the CSLSCQRY return code. SCI return codes are defined in CSLSRR. Possible return codes for CSLSCQRY are described in the following table.

RSNCODE=*symbol*
RSNCODE=*(r1-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. SCI reason codes are defined in CSLSRR. Possible reason codes for CSLSCQRY are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r1-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

SCOPE=*IMSPLEX*
SCOPE=*LOCAL*
SCOPE=*TYPE*
(Optional) - Specifies the scope of information that is being requested.

IMSPLEX
This option returns data for all of the members in the IMSplex.

LOCAL
This option returns information for all of the members on the local z/OS image.

TYPE
This option returns information for all of the members that are of the specified IMSplex member type (and optionally subtype).

SUBTYPE=*symbol*
SUBTYPE=*(r1-r12)*
(Optional) - Four-byte input parameter that specifies the address of an 8-byte subtype that further qualifies the IMSplex member type about which information is being requested. This subtype is defined by the IMSplex member and was specified on the CSLSCREG request.

This parameter is valid only when SCOPE=TYPE.

TYPE=*symbol*
TYPE=*'AOP'*
TYPE=*'BATCH'*
TYPE=*'CQS'*
TYPE=*'DBRC'*
TYPE=*'IMS'*
TYPE=*'IMSCON'*
TYPE=*'ODBM'*
TYPE=*'OM'*
TYPE=*'OTHER'*
TYPE=*'RM'*
TYPE=*'SCI'*
(Optional) - Specifies the IMSplex member type for which the query is being issued. SCI will return information for all of the members that are of the specified IMSplex member type (and, optionally, subtype). This parameter is required when SCOPE=TYPE.

If this parameter is passed as a literal, the literal must be enclosed in single quotation marks. If it is passed as a symbol, the symbol points to a word in storage that contains the code for the member type. If it is passed as a register, the register contains the member type code in the low-order half word of the register.

The code for the member type can be obtained by using the CSLSTPIX macro.

CSLSCQRY return and reason codes

The following table lists the return and reason codes that can be returned on a CSLSCQRY macro request. Also included is the meaning of a reason code (that is, what possibly caused it). In addition, CSLSCQRY can return any of the return codes listed in the following table.

Table 61. CSLSCQRY return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'01000008'	X'00002050'	The caller of the service attempted to pass an invalid parameter list. The request is rejected.
X'0100000C'	X'00003004'	No member data was returned for the request.
X'01000014'	X'00005048'	SCI was unable to obtain storage for the output area of the request.

Related reference
“Writing a CSL SCI client” on page 199
You must establish a connection to SCI (Structured Call Interface) in order to write a program that participates in an IMSplex (such as an AOP) and allows your IMSplex member to communicate with

other IMSplex members. Without a connection to SCI, a program cannot participate in an IMSplex and communicate with other IMSplex members.

"CSLSCREG: registration request" on page 215
The Structured Call Interface (SCI) registration request is used to create a connection between an IMSplex member and SCI. Before SCI can be used for communication within the IMSplex, an IMSplex member must issue the CSLSCREG request and receive an SCI token when the request completes in order to be recognized by SCI.

CSLSCQSC: quiesce request

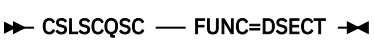
The SCI Quiesce request tells SCI to stop routing messages and requests that have been routed by TYPE to the issuing IMSplex member. After this request has successfully completed, the only messages and requests that are routed to the member are those that are routed directly by SCITOKEN or by NAME.

Note: Because of the asynchronous nature of the processes within the IMSplex and z/OS, messages and requests routed by TYPE might still be received by the IMSplex member after successful completion of the CSLSCQSC FUNC=QUIESCE request. The potential for this occurring is small, but it can happen. The IMSplex member must be able to handle a message or request coming in after the CSLSCQSC FUNC=QUIESCE has successfully completed.

CSLSCQSC syntax

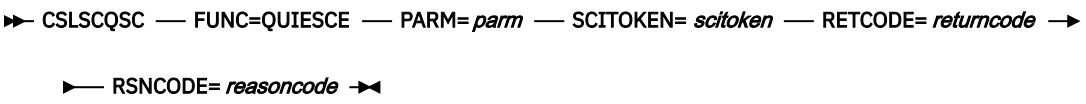
CSLSCQSC DSECT syntax

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



CSLSCQSC QUIESCE syntax

The CSLSCQSC FUNC=QUIESCE request quiesces the connection between SCI and the IMSplex member. After the successful completion of the request, only messages and requests that are routed directly by SCITOKEN or by NAME are sent to this IMSplex member.



CSLSCQSC parameters

PARM=*symbol*
PARM=*(r1-r12)*
(Required) - Specifies the CSLSCQSC parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by SQSC_PARMLN.

RETCODE=*symbol*
RETCODE=*(r1-r12)*
(Required) - Specifies a 4-byte field to receive the return code on output. SCI return codes are defined in CSLSRR. Possible return codes for CSLSCQSC are described in the following table.

RSNCODE=*symbol*
RSNCODE=*(r1-r12)*
(Required) - Specifies a 4-byte field to receive the reason code on output. SCI reason codes are defined in CSLSRR. Possible reason codes for CSLSCQSC are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r1-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

CSLSCQSC return and reason codes

The following table lists the return and reason codes that can be returned on a CSLSCQSC macro request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 62. CSLSCQSC return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'01000008'	X'01002038'	The parameter list version is invalid.
X'01000010'	X'00004000'	SCI is not active.
	X'00004FFF'	The function is not supported.
X'01000014'	X'00005000'	An SCI internal error occurred.

Related reference

"CSLSCMSG: send message request" on page 204
By issuing the CSLSCMSG request, you can send a message to one or more other IMSplex members. The target members are specified by SCITOKEN, member name, or member type.

CSLSCRDY: ready request

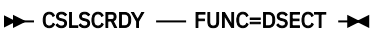
The SCI ready request enables the IMSplex member to receive messages and requests that are routed by TYPE. After the CSLSCREG request is issued and until CSLSCRDY is issued, the IMSplex member can only receive requests that are routed directly to a single target address space. The IMSplex member can send messages and requests that are routed by any method.

Note: The IMSplex member must be ready to process messages and requests that have been routed by TYPE when CSLSCRDY is issued. Because of the asynchronous nature of an IMSplex, the member might receive a message or request that has been routed by TYPE before control is returned after issuing CSLSCRDY.

CSLSCRDY syntax

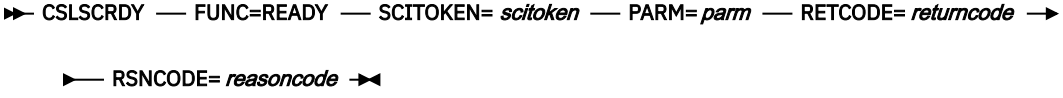
DSECT syntax

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



READY syntax

The CSLSCRDY FUNC=READY request tells SCI that the IMSplex member is now ready to receive messages and requests that are routed by an IMSplex member type.



CSLSCRDY parameters

PARM=*symbol*
PARM=*(r1-r12)*
(Required) - Specifies the CSLSCRDY parameter list. The length of the parameter list must be equal to the parameter list length EQU value defined by SRDY_PARMLN.

RETCODE=*symbol*
RETCODE=*(r1-r12)*
(Required) - Specifies a 4-byte field to receive the return code on output. SCI return codes are defined in CSLSRR. Possible reason codes for CSLSCRDY are described in the following table.

RSNCODE=*symbol*
RSNCODE=*(r1-r12)*
(Required) - Specifies the address of a 4-byte field to receive the CSLSCRDY reason code. SCI reason codes are defined in CSLSRR. Possible reason codes for CSLSCRDY are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r1-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

CSLSCRDY return and reason codes

The following table lists the return and reason codes that can be returned on a CSLSCRDY macro request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 63. CSLSCRDY return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
	X'00002038'	The parameter list version is invalid.
X'01000010'	X'00004000'	SCI is not active.
	X'00004FFF'	The function is not supported.
X'01000014'	X'00005000'	An SCI internal error occurred.

Related reference

"CSLSCMSG: send message request" on page 204
By issuing the CSLSCMSG request, you can send a message to one or more other IMSplex members. The target members are specified by SCITOKEN, member name, or member type.

CSLSCREG: registration request

The Structured Call Interface (SCI) registration request is used to create a connection between an IMSplex member and SCI. Before SCI can be used for communication within the IMSplex, an IMSplex member must issue the CSLSCREG request and receive an SCI token when the request completes in order to be recognized by SCI.

This token is used with all subsequent SCI requests. If SCI terminates while the IMSplex member is active, the member is still registered when SCI becomes active again. The SCI token that the member received on the initial CSLSCREG request is still valid.

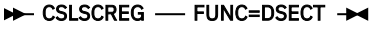
Restrictions:

- CSLSCREG is not supported when the caller's address space has been marked non-swappable by a SYSEVENT DONTSWAP call. Issuing a CSLSCREG in this environment produces unpredictable results. A caller that issued a SYSEVENT DONTSWAP must issue a SYSEVENT OKSWAP before registering with SCI.
- A single address space may register with SCI more than once. However, all registrations from a single address space must be made in the same PSW key and state (supervisor or problem) as the first active registration.

CSLSCREG syntax

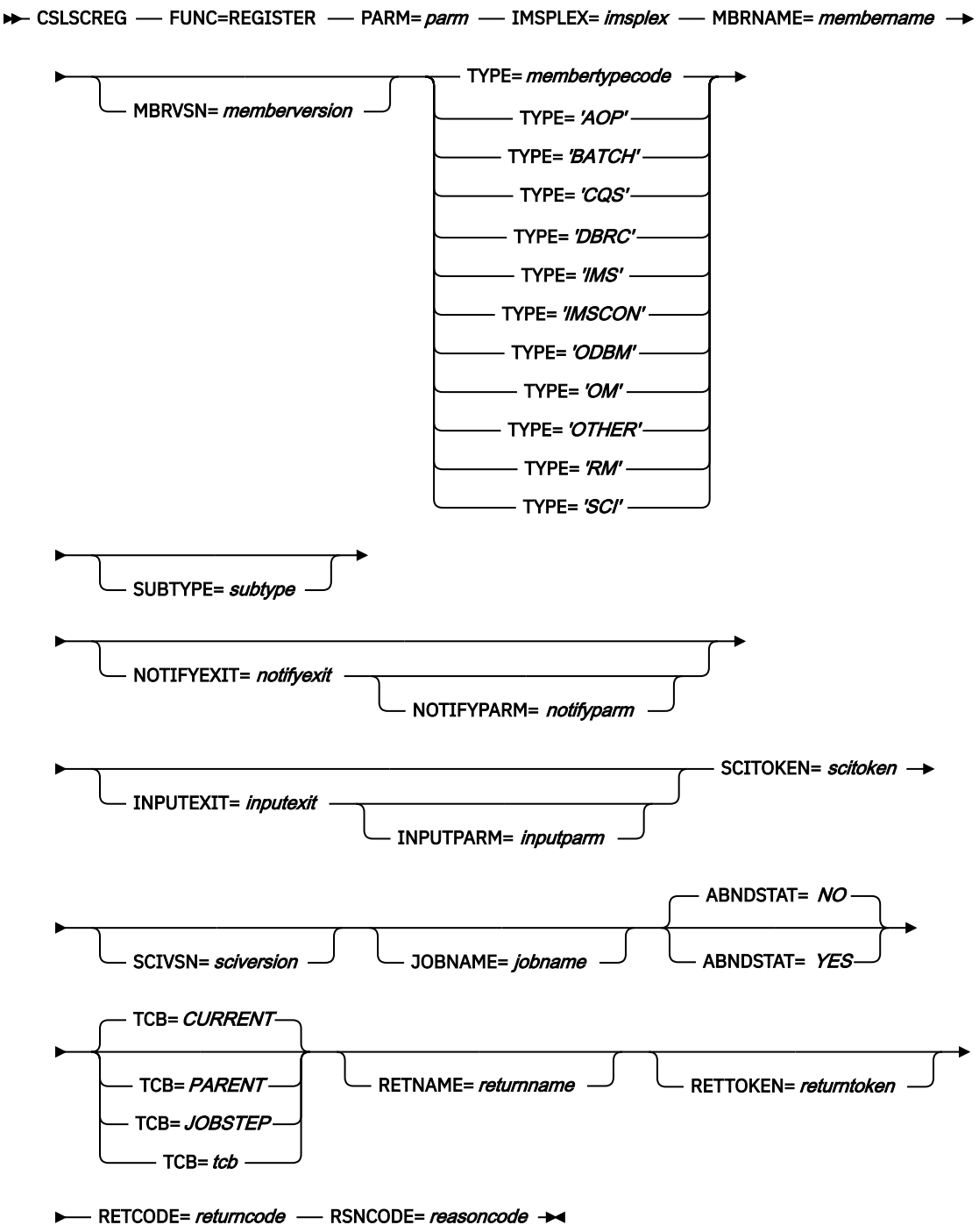
CSLSCREG DSECT syntax

Use the DSECT function of a CSLSCREG request to include equate (EQU) statements in your program for the CSLSCREG parameter list length, the IMSplex types and the CSLSCREG return and reason codes.



CSLSCREG REGISTER syntax

The CSLSCREG FUNC=REGISTER request establishes a connection between an IMSplex member and SCI. An SCI token is returned on successful completion of this request. This token must be used on all subsequent SCI requests. Until the CSLSCRDY FUNC=READY request is issued, the IMSplex member receives only messages and requests that are routed directly to it (by SCITOKEN or by NAME). Messages and requests that are routed by TYPE are not routed to this member. Messages and requests routed by any method can be sent by this member when the CSLSCREG FUNC=READY request has been successfully completed. The syntax for the REGISTER function of the CSLSCREG request follows.



CSLSCREG parameters

ABNDSTAT=*NO*
ABNDSTAT=*YES*

(Optional) - Indicates if SCI is to keep track of the member if the member abnormally terminates. If ABNDSTAT=YES is specified, SCI will retain an entry for the member with a status of ABTERM. If the member normally terminates or if the member abnormally terminates after a successful CSLSCDRG, SCI does not keep a record of the member.

This parameter is ignored for non-authorized IMSplex members.

IMSPLEX=*symbol*
IMSPLEX=*(r2-r12)*
(Required) - Specifies the address of a 1- to 5-character IMSplex name. The IMSPlex name identifies the SCI to which this request is directed. If specified as a symbol, the symbol references storage that contains the IMSplex name.

INPUTEXIT=*symbol*
INPUTEXIT=*(r2-r12)*
(Optional) - Specifies the address of the SCI input exit routine. The input exit is called each time there is a message or request for the member.

INPUTPARM=*symbol*
INPUTPARM=*(r2-r12)*
(Optional) - Specifies the address of an 8-byte area that contains member data. This data is passed to the input exit routine each time it is called. If specified as a symbol, the symbol references storage that contains the member data.

JOBNAME=*symbol*
JOBNAME=*(r2-r12)*
(Optional) - Specifies the address of an 8-byte area to receive the SCI jobname.

MBRNAME=*symbol*
MBRNAME=*(r2-r12)*
(Required) - Specifies the address of an 8-byte name of the IMSplex member registering with SCI. For an authorized member, this name must be unique within the IMSplex. For a non-authorized member, this name does not need to be unique. If it is specified as a symbol, the symbol refers to storage that contains the IMSplex member name. Valid characters for the name are A-Z, 0-9, and special characters @, #, and $.

MBRVSN=*symbol*
MBRVSN=*(r2-r12)*
(Optional) - Specifies the address of a 4-byte version of the IMSplex member registering with SCI. This version number is passed in the parameter list of the SCI Notify exit when this IMSplex member is the subject of the event. It is also passed in the parameter list of the SCI Input exit for messages and requests that originate from this member. If MBRVSN is not specified, the version number in the exit parameter list is set to zeros. If it is specified as a symbol, the symbol references storage that contains the IMSplex member version.

SCI does not validate this field; however, the field can be output on the QRY IMSPLEX command. It is assumed to have the following format: X'00vvrrmm'.

- 00 - this byte is ignored
- vv - version number
- rr - release number
- mm - modification level or subrelease number

For example, X'00080100' would be output as 8.1.0.

NOTIFYEXIT=*symbol*
NOTIFYEXIT=*(r2-r12)*
(Optional) - Specifies the address of the SCI Notify exit routine. The Notify exit is driven whenever there is a change of status of an IMSplex member.

NOTIFYPARM=*symbol*
NOTIFYPARM=*(r2-r12)*
(Optional) - Specifies the address of an 8-byte area that contains member data. This data is passed to the Notify exit routine each time it is called. If it is specified as a symbol, the symbol references storage that contains the member data.

PARM=*symbol*
PARM=*(r2-r12)*
(Required) - Specifies the address of a parameter list used by the request to pass the parameters to SCI. The length of the storage must be at least equal to the value of SREG_LN.

RETCODE=*symbol*
RETCODE=*(r2-r12)*
(Required) - Specifies the address of a 4-byte field to receive the CSLSCREG return code. SCI return codes are defined in CSLSRR. Possible return codes for CSLSCREG are described in the following table.

RETNAME=*symbol*
RETNAME=*(r2-r12)*
(Optional) - Specifies the address of an 8-byte field to receive the name of the SCI that processes the registration request.

RETTOKEN=*symbol*
RETTOKEN=*(r2-r12)*
(Optional) - Specifies the address of a 16-byte field to receive the SCI token of the SCI that processes the registration request.

RSNCODE=*symbol*
RSNCODE=*(r2-r12)*
(Required) - Specifies the address of a 4-byte field to receive the CSLSCREG reason code. SCI reason codes are defined in CSLSRR. Possible reason codes for CSLSCREG are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r2-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

SCIVSN=*symbol*
SCIVSN=*(r2-r12)*
(Optional) - Specifies the address of a 4-byte field to receive the SCI 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 subrelease number.

Example: SCI version 1.1.0 is shown as X'00010100'.

SUBTYPE=*symbol*
SUBTYPE=*(r2-r12)*
(Optional) - Specifies the address of the 8-byte subtype of the member registering with SCI. The subtype is defined by the user and can be any eight characters. If it is specified as a symbol, the symbol references storage that contains the subtype. If not specified, this parameter is set to blanks. If SUBTYPE is not specified, it will be set to blanks.

The subtype can contain alphanumeric characters (A-Z, 0-9), or printable characters (not case sensitive), with the exception of the characters &, <, and >. OM converts any invalid data placed in this field to periods (.) before sending the XML output to the client.

TCB=*CURRENT*
TCB=*JOBSTEP*
TCB=*PARENT*
TCB=*symbol*
TCB=*(r2-r12)*
(Optional) - Specifies the TCB with which the new SCI connection is associated. The SCI connection persists until one of the following occurs:

- The member deregisters by using CSLSCDRG.

- The TCB associated with the connection terminates.

All callers of CSLSCREG can specify the following values for the TCB parameter:

CURRENT
Associates the SCI connection with the currently executing TCB. This is the default.

JOBSTEP
Associates the SCI connection with the JOBSTEP TCB of the TCB under which the CSLSCREG request is issued. This TCB is specified by TCBJSTCB.

PARENT
Associates the SCI connection with the TCB that attached the currently-executing TCB.

For non-authorized callers, the indicated TCB must have the same storage key associated with it as the caller's current PSW key (that is, TCBPKF must match the current PSW key).

Authorized callers can, in addition, identify an explicit TCB by specifying a symbol or register. If specified as a symbol, the symbol must be the label on a word of storage containing the address of the TCB. If specified as a register, the register must contain the TCB address.

TYPE=*membertypecode*
TYPE=*'AOP'*
TYPE=*'BATCH'*
TYPE=*'CQS'*
TYPE=*'DBRC'*
TYPE=*'IMS'*
TYPE=*'IMSCON'*
TYPE=*'ODBM'*
TYPE=*'OM'*
TYPE=*'OTHER'*
TYPE=*'RM'*
TYPE=*'SCI'*
(Required) - Specifies the SCI member type of the address space that is registering with SCI.

If this parameter is passed as a literal, the literal must be enclosed in single quotation marks. If this parameter is passed as a symbol or register, the symbol or register must contain the member type code.

The code for the member type can be obtained by using the CSLSTPIX macro. Member types include:

AOP
This SCI type is an automated operator program. It interacts with OM by sending commands and receiving responses to the commands.

Batch
This SCI type is an IMS batch region. It interacts as an IMS DL/I batch or utility region.

CQS
This SCI type is an IMS Common Queue Server. It provides access to a set of common queues within the IMSplex.

DBRC
This SCI type is an IMS Database Recovery Control Region.

IMS
This SCI type is an IMS region. It can include the database manager, transaction manager, and FDBR (an IMS control region that recovers database resources when an IMS database manager fails). SUBTYPE is used to further qualify a particular control region (for example, DBDC, DBCTL, DCCTL, or FDBR).

IMSCON
This SCI type is a connector to IMS. It acts as an interface between IMS and protocols that are not supported by IMS directly (such as TCP/IP).

ODBM
This SCI type is an IMS Open Database Manager, which is part of the CSL. It receives requests to access and manipulate IMS databases from clients, such as IMS Connect or an ODBA application, and routes the requests to the IMS DB systems in the IMSplex that manage the database. When ODBM is used in support of IMS Connect and the IMS Universal drivers, ODBM translates the incoming database access requests from the low-level DRDA protocol into the DL/I calls used by IMS and back again on output.

OM
This SCI type is an IMS Operations Manager, which is part of the CSL. It receives commands from AOPs, routes the commands to other members of the IMSplex that have registered for the command, consolidates the responses to the command, and sends the output back to the originating AOP.

Other
This SCI type is any other address space that does not fall into one of the defined SCI types.

RM
This SCI type is an IMS Resource Manager, which is part of the CSL. It manages resources within the IMSplex and coordinates IMSplex-wide processes. SUBTYPE is used to further qualify whether there is a single RM in the IMSplex (SNGLRM) or there are multiple RMs in the IMSplex (MULTRM).

SCI
This SCI type is an IMS SCI, which is part of the CSL. It manages communications within the IMSplex.

CSLSCREG return and reason codes

The following table lists the return and reason codes that can be returned on a CSLSCREG macro request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 64. CSLSCREG return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'01000004'	X'00001000'	The member is already registered. If the member is authorized, the member name is already registered to this SCI and the SCI token is returned. If the member is not authorized, there are three registrations from the current TCB and no more registrations are allowed. One of the SCI tokens is returned.
X'01000008'	X'00002010'	An invalid type was passed
	X'00002038'	The parameter list version is invalid.
X'01000010'	X'00004000'	SCI is not active.
	X'00004004'	CSLSRG00 could not be loaded.
	X'00004008'	The user ID of the member address space is not authorized to register with this SCI.
	X'00004010'	The member name, *membername*, is not unique for the authorized client. The registration is rejected.
	X'00004028'	A non-authorized member tried to register as an authorized system SCI type.
	X'0000402C'	Either the caller key or state does not match the key or state of the existing registration.
	X'00004FFF'	The function is not supported.
X'01000014'	X'00005000'	An SCI internal error occurred.

Table 64. CSLSCREG return and reason codes (continued)

Return code	Reason code	Meaning
	X'00005004'	An ESTAE add error occurred.
	X'00005008'	A BPE SVC error occurred.
	X'0000500C'	A z/OS Name/Token retrieve error occurred.
	X'00005010'	An error occurred while establishing ResMgr.
	X'00005014'	An error occurred while obtaining storage.
	X'00005018'	An error occurred while obtaining a TTOKEN.
	X'0000501C'	An ALESERV error occurred.
	X'00005020'	An ENQ resource error occurred.
	X'00005050'	A BPECGBET error occurred in CSLSRGS0.
	X'00005054'	An ALESERV error occurred in CSLSRGS0.
	X'00005058'	A BPEHTADD error occurred in CSLSRGS0.
	X'00005064'	A BPEHTFND token error occurred in CSLSRGS0.
	X'00005070'	The SCI buffer manager could not be initialized.
	X'00005080'	The z/OS cross-system coupling facility join for the member failed.
	X'00005084'	A non-authorized member specified an explicit connection TCB.
	X'00005088'	The connection TCB key does not match the CSLSCREG caller's key.
	X'0000508C'	The TCB type code passed on the CSLSCREG request is invalid.
	X'00005090'	Error enqueuing registration AWE. This is an internal error.
	X'00005094'	Error scheduling SRB to SCI. This is an internal error.
	X'00005500'	An abend occurred during CSLSCREG processing.

Related concepts

"Issue CSL RM requests to manage global resources" on page 164
Before a client can access or change global resource information, it must register to SCI using the CSLSCREG request. After the client registers to SCI, it must register to RM using the CSLRMREG request. The client must issue an SCI registration request for every IMSplex with which it intends to communicate.

Related tasks

"Registering an ODBM client" on page 99
To register with ODBM, a client must first register with the CSL SCI and then with all active ODBMs in the IMSplex.

Related reference

"CSLOMCMD: command request" on page 107
By using the CSLOMCMD request, your AOP client application that is running on the host can issue requests and send commands to OM.

"CSLOMI: API request" on page 111
With the CSLOMI request, your AOP client can communicate with a z/OS address space that acts as an OM AOP client. You can then issue OM requests and send QUERY commands to OM.

"CSLOMQRY: query request" on page 121

With the CSLOMQRY request, any AOP client that is running on the host can request OM-specific information.

“CSLSCQRY: query request” on page 210
By issuing the CSLSCQRY request, an IMSplex member can obtain information about the members of the IMSplex.

CSL SCI Notify Client exit routine (Exit Routines)

“CSLSCRQS: send request” on page 225
By issuing the CSLSCRQS request, an IMSplex member can send a request to another member in the IMSplex. The target member can be specified by SCITOKEN, member name, or member type.

CSLSCRQR request return request

Issuing the CSLSCRQR request returns a request to the IMSplex member from which the request originated. The return request should be issued when the server has completed the request and is ready to return the output from the request.

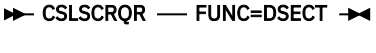
CSLSCRQR returns a request to the IMSplex member from which the request originated. It should be issued when the server has completed the request and is ready to return the output from the request. It copies the output back to the requestor's address space.

Only request servers can issue CSLSCRQR because an IMSplex member cannot issue the macro without first receiving a request. A request server must be authorized and running key 7.

CSLSCRQR syntax

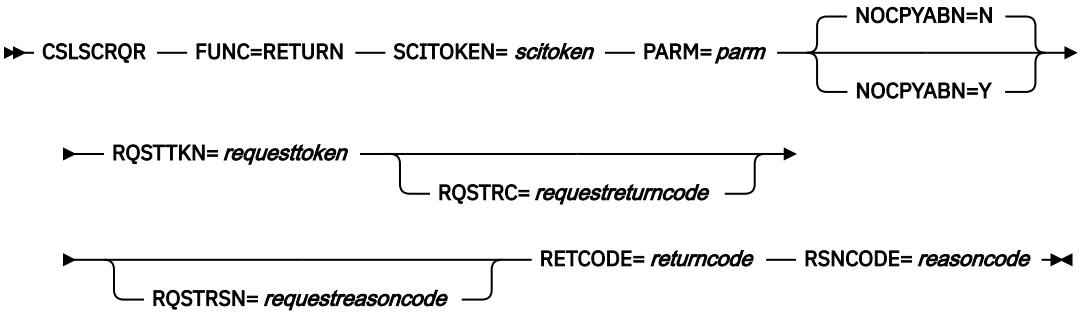
CSLSCRQR DSECT syntax

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



CSLSCRQR RETURN syntax

The syntax for the CSLSCRQR FUNC=RETURN request follows.



CSLSCRQR parameters

PARM=*symbol*
PARM=*(r1-r12)*
(Required) - Specifies the CSLSCRQR parameter list. The length of the parameter list must be at least equal to the basic parameter list length of SRQR_PARMLN. However, if you code certain parameters on CSLSCRQR, you must provide a longer parameter list. Use the following table to decide if you must provide a parameter list of a longer length. If you are using more than one parameter listed, and the parameters have different minimum length values, always use the one that ends with the largest numeric value.

Table 65. Parameters that require a parameter list length larger than SRQR_PARMLN

Length EQU	Parameters that require this length
SRQR_PRMLV2	NOCPYABN

You can also define your parameter list to have a length of SRQR_PRMLMX bytes. This EQU is set to the length of the longest CSLSCRQR parameter list version, and the parameter list will always be long enough for any combination of macro parameters. However, this maximum length might change because of maintenance or across IMS releases.

NOCPYABN=N | Y
(Optional) - Specifies how SCI should handle an abend during the copy phase of the request return process.

N
SCI will take a dump and issue a message as determined by the logic in the ARR or FRR. NOCPYABDN=N is the default.

Y
SCI will not take a dump or issue a message if an abend is encountered while it attempts to copy data back to the requestor. SCI will only write an entry to the LOGREC data set.

This parameter was added as part of the Version 2 parameter list. If you include this parameter, your parameter list must be at least equal to the value specified by SQRQ_PRMLV2.

RQSTRC=*symbol*
RQSTRC=*(r1-r12)*
(Optional) - Specifies the return code that is associated with the request being returned. This return code will be given to the requesting member in the storage pointed to by the RETCODE parameter of the CSLSCRQS that originated this request. If this parameter is not specified, a return code of zero will be given to the requesting member.

If specified as a symbol, the symbol references storage that contains the return code.

RQSTRSN=*symbol*
RQSTRSN=*(r1-r12)*
(Optional) - Specifies the reason code that is associated with the request being returned. This reason code will be given to the requesting member in the storage pointed to by the RSNCODE parameter of the CSLSCRQS that originated this request. If this parameter is not specified, a reason code of zero will be given to the requesting member.

If specified as a symbol, the symbol references storage that contains the return code.

RQSTTKN=*symbol*
RQSTTKN=*(r1-r12)*
(Required) - Specifies the request token that is associated with the request being returned. This request token can be obtained from the input exit parameter list (INXP_RQSTTKN) when the request was presented to the request processing member.

If specified as a symbol, the symbol references storage that contains the return code.

RETCODE=*symbol*
RETCODE=*(r1-r12)*
(Required) - Specifies the address of a 4-byte field to receive the CSLSCRQR return code. SCI return codes are defined in CSLSRR. Possible return codes for CSLSCRQR are described in the following table.

RSNCODE=*symbol*
RSNCODE=*(r1-r12)*
(Required) - Specifies the address of a 4-byte field to receive the CSLSCRQR reason code. SCI reason codes are defined in CSLSRR. Possible reason codes for CSLSCRQR are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r1-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

CSLSCRQR return and reason codes

The following table lists the return and reason codes that can be returned on a CSLSCRQR macro request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 66. CSLSCRQR return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'01000008'	X'00002004'	The function passed to the SCI interface PC routine was invalid.
	X'00002018'	The SCI token is invalid.
	X'00002038'	The parameter list version is invalid.
X'01000010'	X'00004000'	SCI is not active.
	X'0000400C'	The target member is not active.
	X'00004FFF'	The function is not supported.
X'01000014'	X'00005000'	An SCI internal error occurred.
	X'0000502C'	The member could not be found due to an internal BPE hash table services error.
	X'00005030'	An SCI buffer could not be obtained.
	X'00005034'	A key 7 buffer in the SCI address space could not be obtained for a copy of PHDR and parameters.
	X'00005038'	An IEAMSCHD error occurred; SRB could not be scheduled to the target address space.
	X'00005040'	The request is not outstanding and cannot be returned.
	X'00005044'	An SCI-allocated output buffer could not be obtained.
	X'00005500'	An abend occurred during the processing of an SCI request.
	X'00005504'	An abend occurred when the member parameters were copied to the target address space.
	X'00005508'	An abend occurred when the member parameters were copied to the target address space, but the dump and message were suppressed because NOCPYABN=Y was specified.

CSLSCRQS: send request

By issuing the CSLSCRQS request, an IMSplex member can send a request to another member in the IMSplex. The target member can be specified by SCITOKEN, member name, or member type.

A request in an IMSplex can contain both input and output data (from the target member's perspective). This contrasts to a message that can only contain input data (again, from the target member's perspective). The data of a request is copied to the target member's address space. The function is processed, and the output is returned to the requestor's address space. If the request included an ECB, control is returned to the requesting module after the request has been processed by SCI. The requestor must then wait on the ECB.

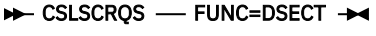
The ECB is posted when the request processing has completed. The requestor then looks at the RETCODE and RSNCODE fields to determine the outcome of the request. If no ECB is included in the request, the RETCODE and RSNCODE fields can be used to determine the outcome of the request when the requesting module gets control back from SCI.

Note: Before issuing CSLSCRQS, the requester should clear the fields that will receive the address and length of the SCI Allocated Output parameters. If the request is not sent to the destination because of an error, or if there is no data to output, SCI will not update the length and address fields.

CSLSCRQS syntax

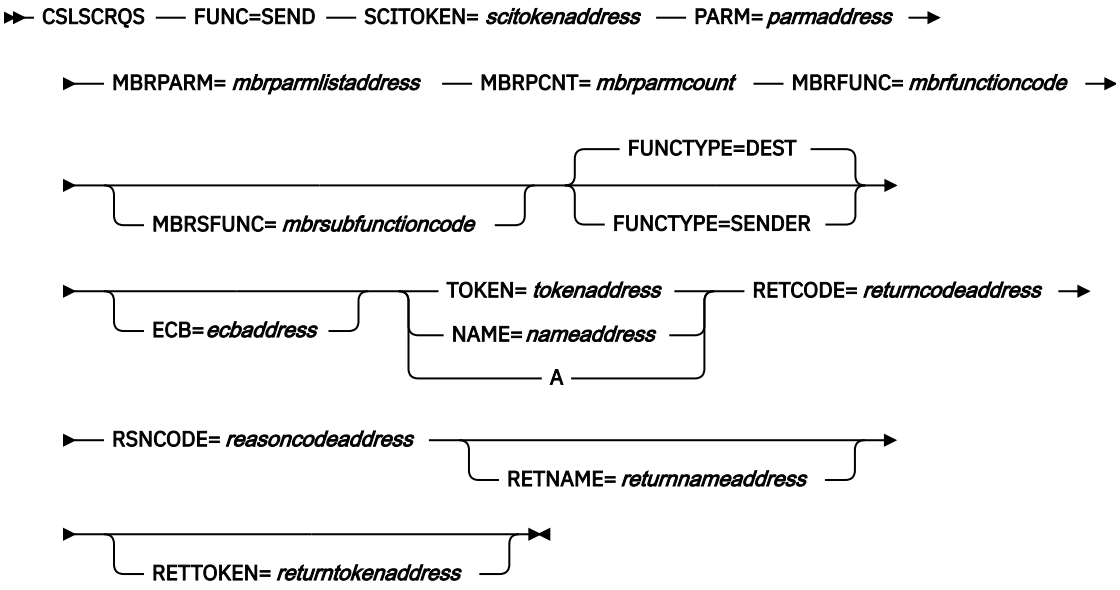
DSECT syntax

Use the DSECT function of a CSLSCRQS request to include equate (EQU) statements in your program for the CSLSCRQS parameter list length, the IMSplex types and the CSLSCRQS return and reason codes.

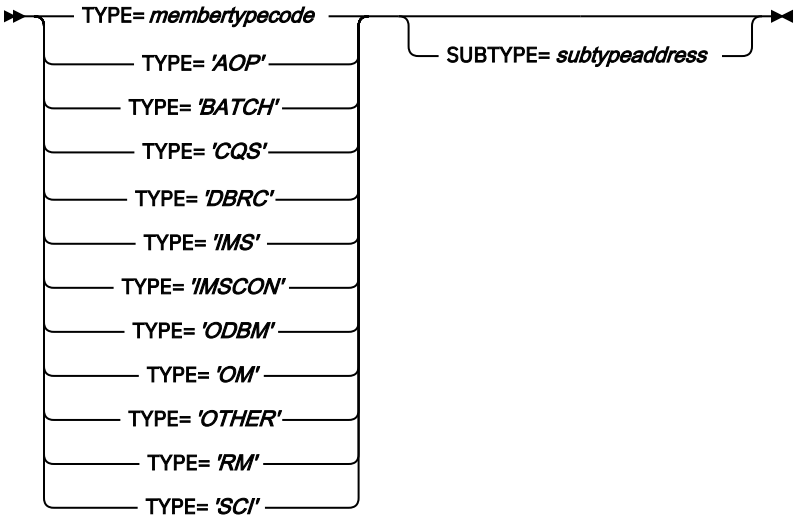


SEND REQUEST syntax

The syntax for the CSLSCRQS FUNC=SEND request follows.



A



CSLSCRQS parameters

ECB=*symbol*
ECB=*(r1-r12)*
(Optional) - Specifies the address of a z/OS ECB used for asynchronous requests. When the request is complete, the ECB specified is posted. If an ECB is not specified, the task is suspended until the request is complete. If an ECB is specified, the invoker of the macro must issue a WAIT (or equivalent) after receiving control from CSLSCRQS, before using or examining any data returned by this macro (including the RETCODE and RSNCODE fields).

FUNCTYPE=DEST
FUNCTYPE=SENDER
(Optional) - Specifies that the MBRFUNC and MBRSFUNC are defined by the DEST (destination) of this request or the SENDER of the request. This indicator is passed to the recipient of the request in the SCI Input exit parameter list.

MBRFUNC= *symbol*
MBRFUNC= *(r1-r12)*
(Required) - Specifies a 4-byte member function code that is passed to the destination of the request in the SCI Input exit parameter list. This function code, along with the MBRSFUNC, identifies the request that is being sent.

If MBRFUNC is a symbol, the symbol points to a four-byte area of storage that contains the function code.

MBRPARM= *symbol*
MBRPARM= *(r1-r12)*
(Required) - Specifies the address of a pre-built parameter list. This parameter list must be built by the requesting module and consists of sets of triplets. Each triplet describes a single parameter in the member parameter list and consists of:

parameterlength
Four-byte parameter that specifies the length of the member parameter.

parameteraddress
Four-byte parameter that specifies the address of the member parameter.

datatype
Four-byte parameter that specifies how this parameter is to be handled by SCI. Equates are provided for each type (included with CSLSCODE). These equates can be used to set the value of data type. Possible values are:

IN
The parameter is an input parameter. It is copied to the destination address space with the request.

OUT
The parameter is an output parameter. It is copied back to the requesting address space when the request is completed by the server. The storage for the parameter must be allocated before the request is issued.

IO
The parameter is both an input and an output parameter. It is copied to the target address space with the request and it is copied back to the requesting address space when the request is complete.

SCI
The parameter is an SCI allocated output parameter. The storage for the parameter is allocated in the requestor's address space when the request is complete. The address of the storage will be returned in the parameter address field and the length will be returned in the parameter length field. The storage must be released by the requestor using the CSLSCBFR request. The eight bytes immediately in front of the address returned for an SCI-allocated output parameter are available for use by the requestor. These eight bytes are not cleared, and might contain residual data from a prior use of the buffer.

The two methods for passing parameters in a parameter list are *by address* and *by value*. Both of these methods can be used when passing parameters in a CSLSCRQS request. The triplet must be setup so that SCI will handle the parameter properly.

- By address

  To pass a parameter by address, the address of the parameter must be passed in *parameteraddress* and the length of the parameter must be passed in *parameterlength*. SCI will get the parameter from *parameteraddress* for data type IN and IO and will store the parameter at *parameteraddress* for data type OUT and IO. The address at which the parameter is stored and its length is returned for data type SCI.

- By value

  To pass a parameter by value, the parameter must be passed in *parameteraddress* and zero must be passed in *parameterlength*. When the length is zero, SCI will copy the value contained in *parameteraddress* to the destination for data type IN. All other data types must be passed by address since SCI requires an address to store any output parameters.

Member Parameter List: The user parameters specified here are presented to the program that receives the request in the member parameter list, the address of which is contained in the Input Exit Parm area field INXP_MBRPLPTR. Each parameter is represented by eight bytes, the first four bytes contain *parameterlength* and the second four bytes contain *parameteraddress* (if *parameteraddress* is an address, the second four bytes point to storage in the local address space, not the requesting address space). If the parameter's data type is SCI, the first four bytes will contain a length of four and the second word's value is unpredictable.

Null Parameters: In some cases a request processing module expects a set number of parameters with a defined order. If a request is to be sent that does not contain all the parameters, null parameters must be sent to ensure the data buffer contains everything that is expected. Null parameters can be sent by specifying zero for *parameterlength* and *parameteraddress*. The eights bytes that represent the parameter in the data buffer will contain zeros. This is true for any data type (IN, OUT, IO or SCI) or method of passing parameters (by address or by value).

MBRPCNT=*symbol*
MBRPCNT=*(r1-r12)*
(Required) - Specifies a 4-byte field that contains the number of member parameters that are included in MBRPARM.

MBRSFUNC=*symbol*
MBRSFUNC=*(r1-r12)*=
(Optional) - Specifies a 4-byte member subfunction code that is passed to the destination of the request in the SCI input exit parameter list. This subfunction code, along with the MBRFUNC, identifies the request that is being sent.

If MBRSFUNC is a symbol, the symbol points to a 4-byte area of storage that contains the sub-function code.

NAME=*symbol*
NAME=*(r1-r12)*
(Optional) - Specifies the address of an 8-byte member name of the destination of this request. This name can be obtained from the Notify exit (when the member joins the IMSplex) or by issuing a CSLSCQRY request.

Note: One of the routing parameters (NAME, TOKEN or TYPE) must be included.

PARM=*symbol*
PARM=*(r1-r12)*
(Required) - Specifies the address of a parameter list used by the request to pass the parameters to SCI. The length of the storage must be at least equal to the value of SRQS_LN.

RETCODE=*symbol*
RETCODE=*(r1-r12)*
(Required) - Specifies the address of a 4-byte field to receive the CSLSCRQS return code. SCI return codes are defined in CSLSRR. Possible return codes for CSLSCRQS are described in the following table.

RETNAME=*symbol*
RETNAME=*(r1-r12)*
(Optional) - Specifies the address of an 8-byte field to receive the name of the SCI that processes the request.

RETTOKEN=*symbol*
RETTOKEN=*(r1-r12)*
(Optional) - Specifies the address of a 16-byte field to receive the SCI token of the SCI that processes the request.

RSNCODE=*symbol*
RSNCODE=*(r1-r12)*
(Required) - Specifies the address of a 4-byte field to receive the CSLSCRQS reason code. SCI reason codes are defined in CSLSRR. Possible reason codes for CSLSCRQS are described in the following table.

SCITOKEN=*symbol*
SCITOKEN=*(r1-r12)*
(Required) - Specifies a 16-byte field containing the SCI token. This token uniquely identifies this connection to SCI. The SCI token was returned by a successful CSLSCREG FUNC=REGISTER request.

TOKEN=*symbol*
TOKEN=*(r1-r12)*
(Optional) - Specifies the address of the 16-byte SCI token of the destination of this request. This token can be obtained either from the Notify exit (when the member joins the IMSplex) or by issuing a SLSCQRY message.

Note: One of the routing parameters (NAME, TOKEN, TYPE) must be included.

TYPE=*symbol*
TYPE=*(r1-r12)*
TYPE=*'AOP'*
TYPE=*'BATCH'*
TYPE=*'CQS'*
TYPE=*'DBRC'*
TYPE=*'IMS'*
TYPE=*'IMSCON'*
TYPE=*'ODBM'*
TYPE=*'OM'*
TYPE=*'OTHER'*
TYPE=*'RM'*
TYPE=*'SCI'*
Input parameter that specifies the IMSplex member type of the IMSplex member to which this request should be routed. The IMSplex member type routing can be further qualified by using the SUBTYPE parameter. If TYPE is specified, SCI chooses the IMSplex member of the requested type to which the request is sent.

If member type is specified as a literal, the literal must be enclosed in single quotes. If this parameter is passed as a symbol or register, the symbol or register must contain the member type code. The member type code can be obtained by using the CSLSTPIX macro.

Note: One of the routing parameters (NAME, TOKEN, TYPE) must be included.

CSLSCRQS return and reason codes

The following table lists the return and reason codes that can be returned on a CSLSCRQS macro request. Also included is the meaning of a reason code (that is, what possibly caused it).

Table 67. CSLSCRQS return and reason codes

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'01000008'	X'00002004'	The function passed to the SCI interface PC routine is invalid.
	X'00002008'	The number of parameters passed was either less than or equal to zero, or greater than the maximum allowed.
	X'00002010'	An invalid type was passed.
	X'00002018'	This SCI token is invalid.
	X'00002024'	The PHDR length is invalid.
	X'00002028'	The routing data length is invalid.
	X'0000202C'	The request target member is not key 7.
	X'00002030'	The request target member is not authorized.
	X'00002034'	The length of the parameters is too large for a non-authorized caller.
	X'00002038'	The parameter list version is invalid.
	X’0000203C’	CSLSCRQS was called in SRB mode for a synchronous request (no ECB= coded).
X'01000010'	X'00004000'	SCI is not active.
	X'0000400C'	The destination member is not active. The destination member might have been designated by name, token, or type.
	X'0000401C'	The calling member is in the process of deregistering from SCI.
	X'00004020'	The request timed out.
X'01000014'	X'00005000'	An SCI internal error occurred.
	X'00005004'	An ESTAE add error occurred.
	X'00005024'	An SRB routine error occurred.
	X'00005028'	The routing type is invalid.
	X'0000502C'	The member could not be found due to an internal BPE hash table services error.
	X'00005030'	A buffer in the destination member's address space could not be obtained.
	X'00005034'	A key 7 buffer in the SCI address space could not be obtained for a copy of PHDR and parameters.
	X'00005038'	An IEAMSCHD error occurred; an SRB could not be scheduled to the target address space.
	X'0000503C'	MRT could not be expanded.
	X'00005044'	An SCI-allocated output buffer could not be obtained.
	X'0000504C'	A message SRB key 7 parameter area could not be obtained.
	X'0000507C'	An IXCMSGO error occurred.
	X'00005500'	An abend occurred during CSLSCRQS processing.

Table 67. CSLSCRQS return and reason codes (continued)

Return code	Reason code	Meaning
	X'00005504'	An abend occurred while the member parameters were copied to the target address space.

Related reference

“CSLSCREG: registration request” on page 215

The Structured Call Interface (SCI) registration request is used to create a connection between an IMSplex member and SCI. Before SCI can be used for communication within the IMSplex, an IMSplex member must issue the CSLSCREG request and receive an SCI token when the request completes in order to be recognized by SCI.

Chapter 9. CSL Operations Manager XML output

Command responses that are returned through the OM API are embedded in XML tags using codepage 037. XML output is generated for responses to the CSLOMI, CSLOMCMD, and CSLOMQRY requests.

Note: The OM response is intended as a programming interface, not as an interface that produces prebuilt messages to be displayed on a screen. For OM requests, the output is passed back in the OUTPUT= buffer. For messages, the output is returned to the SCI input exit. The OM response is returned encapsulated in XML tags.

Related concepts

"CSL OM automated operator program clients" on page 125
OM provides an API interface for application programs that automate operator actions known as automated operator programs (AOP). You can use an AOP to issue commands that are embedded in an OM API request to an OM.

Related reference

"CSLOMCMD: command request" on page 107
By using the CSLOMCMD request, your AOP client application that is running on the host can issue requests and send commands to OM.

"CSLOMI: API request" on page 111
With the CSLOMI request, your AOP client can communicate with a z/OS address space that acts as an OM AOP client. You can then issue OM requests and send QUERY commands to OM.

"CSLOMQRY: query request" on page 121
With the CSLOMQRY request, any AOP client that is running on the host can request OM-specific information.

"CSLOMRSP: command response request" on page 153
The CSLOMRSP request is issued by a command processing client in response to a command. Command response information is consolidated and sent to OM.

CSLOMI XML output examples

Each of the command syntax examples contain a sample of CSLOMI XML output. The examples present different scenarios that generate XML output based on the commands that are used in the example.

CSLOMI XML output

One or more of the sets of tags in the following output example is returned on each CSLOMI request.

```
<imsout>
   <ctl>
       <omname> </omname>
       <omvsn> </omvsn>
       <xmlvsn> </xmlvsn>
       <statime> </statime>
       <stotime> </stotime>
       <staseq> </staseq>
       <stoseq> </stoseq>
       <rqsttkn1> </rqsttkn1>
       <rqsttkn2> </rqsttkn2>
       <rc> </rc>
       <rsn> </rsn>
            <rsnmsg> </rsnmsg>
  </ctl>
  <cmdclients>
    <mbr name="membername">
      <typ> </typ>
      <styp> </styp>
      <vsn> </vsn>
      <jobname> </jobname>
   </mbr>
  </cmdclients>
```

```
  <cmdsyntax> </cmdsyntax>
  <cmddtd> </cmddtd>
  <cmdtext> </cmdtext>
  <cmderr>
    <mbr name="membername">
      <typ> </typ>
      <styp> </styp>
      <rc> </rc>
      <rsn> </rsn>
   </mbr>
  </cmderr>
  <cmdsecerr>
    <exit>
       <rc> </rc>
       <userdata> </userdata>
    </exit>
    <saf>
       <rc> </rc>
       <racfrc> </racfrc>
       <racfrsn> </racfrsn>
    </saf>
 </cmdsecerr>
 <cmd>
    <master> </master>
    <userid> </userid>
    <verb> </verb>
    <kwd> </kwd>
    <input> </input>
 </cmd>
 <cmdrsphdr>
    <hdr ... />
 </cmdrsphdr>
 <cmdrspdata>
    <rsp> </rsp>
 </cmdrspdata>
 <msgdata>
   <mbr name="membername">
     <msg> </msg>
   </mbr>
 </msgdata>
</imsout>
```

Issue IMS command example

The following examples are CSLOMI XML output. In the following command example, the **QUERY TRAN** command was routed to IMSA with a timeout value of 10 seconds.

```
OM API Input:
CMD(QUERY TRAN) NAME(SKS*)) ROUTE(IMSA) TIMEOUT(10) RQSTTKN2(QTRANCMD)

OM API Output:
<imsout>
  <ctl>
    <omname>OM1</omname>
    <omvsn>1.1.0</omvsn>
    <xmlvsn>1</xmlvsn>
    <statime>1999.341 12:52:44.46</statime>
    <stotime>1999.341 12:52:44.46</stotime>
    <staseq>B342BCC72A34D206</staseq>
    <stoseq>B342BCC75CD52208</stoseq>
    <rqsttkn2>QTRANCMD</rqsttkn2>
    <rc>0</rc> <rsn>0</rsn>
  </ctl>
  <cmd>
    <master>IMS1</master>
    <verb>QRY</verb>
    <kwd>TRAN</kwd>
    <input>QUERY TRAN</input>
  </cmd>
  <cmdrsphdr>
    <hdr slbl="TRAN" llbl="TranCode" scope="LCL" sort="a" key="1" scroll="no"
     len="8" dtype="CHAR" align="left" />
    <hdr slbl="MBR" llbl="MbrName" scope="LCL" sort="a" key="4" scroll="no"
     len="8" dtype="CHAR" align="left" />
    <hdr slbl="CC" llbl="CC" scope="LCL" key="0" scroll="YES" len="4" dtype="INT"
     align="right" />
  </cmdrsphdr>
  <cmdrspdata>
    <rsp> TRAN(SKS1) MBR(IMSA) CC(0) </rsp>
```

```
    <rsp> TRAN(SKS2) MBR(IMSA) CC(0) </rsp>
    <rsp> TRAN(SKS3) MBR(IMSA) CC(0) </rsp>
    <rsp> TRAN(SKS4) MBR(IMSA) CC(0) </rsp>
    <rsp> TRAN(SKS5) MBR(IMSA) CC(0) </rsp>
  </cmdrspdata>
</imsout>
```

Query client list example

In the following client list example, the Operations Manager (OM) returns a list of client names that are currently registered for command processing.

```
OM API Input:
QUERY(CMDCLIENTS) RQSTTKN2(CLIENTLIST)
OM API Output:
<imsout>
  <ctl>
    <omname>OM1</omname>
    <omvsn>1.1.0</omvsn>
    <xmlvsn>1</xmlvsn>
    <statime>1999.341 12:52:44.46</statime>
    <stotime>1999.341 12:52:44.46</stotime>
    <staseq>B342BCC72A34D206</staseq>
    <stoseq>B342BCC75CD52208</stoseq>
    <rqsttkn2>CLIENTLIST</rqsttkn2>
    <rc>0</rc> <rsn>0</rsn>
  </ctl>
  <cmdclients>
    <mbr name=IMSA>
      <typ>DBDC</typ>
      <vsn>0800</vsn>
      <jobname>IMSJOB01</jobname>
    </mbr>
    <mbr name=IMSB>
      <typ>DBDC</typ>
      <vsn>0800</vsn>
      <jobname>IMSJOB02</jobname>
    </mbr>
  </cmdclients>
</imsout>
```

Query command syntax example

The following command syntax example returns the command syntax for currently registered commands. In this example, the **QUERY TRAN** command is the only command that is registered to OM, and the keyword NAME is associated with it.

```
OM API Input:
QUERY(CMDSYNTAX) RQSTTKN2(CMDLIST)
OM API Output:
<imsout>
  <ctl>
    <omname>OM1</omname>
    <omvsn>1.1.0</omvsn>
    <xmlvsn>1</xmlvsn>
    <statime>1999.341 12:52:44.46</statime>
    <stotime>1999.341 12:52:44.46</stotime>
    <staseq>B342BCC72A34D206</staseq>
    <stoseq>B342BCC75CD52208</stoseq>
    <rqsttkn2>CMDLIST</rqsttkn2>
    <rc>0</rc> <rsn>0</rsn>
  </ctl>

  <cmdsyntax>
    <root>
      <resource name="TRAN">
        <verb name="QUERY">
          <keyword name="NAME">
            <var name="tranname*"/>
          </keyword>
        </verb>
      </resource>
    </root>
  </cmdsyntax>
```

```
  <cmdtext>
    NEXT "Next"
    BACK "Back"
    FINISH "Finish"
    CANCEL "Cancel"
    SUMMARY "Summary"
    TRAN_NAME "Transaction"
    TRAN_QUERY_NAME "Query"
    TRAN_QUERY_NAME_NAME "Name"
    TRAN_QUERY_NAME_TEXT "Name of transaction."
    TRAN_QUERY_NAME_VAR "tranname*"
  </cmdtext>
</imsout>
```

Related reference

"XML tags returned as CSL OM responses" on page 238
Different XML tags can be returned as CSL OM responses. Each tag name is delimited by the characters < and >. Tags can be nested within parent tags to encapsulate related information.

QUERY TRAN command (Commands)

CSLOMCMD output

The command syntax example contains a sample of CSLOMCMD XML output. The example presents a scenario that generates XML output based on the commands that are used in the example.

CSLOMCMD XML output

The tags in the following output example can be returned as a result of a CSLOMCMD request.

```
<?xml version="1.0"?>
<!DOCTYPE imsout SYSTEM "imsout.dtd">
<imsout>
   <ctl>
       <omname> </omname>
       <omvsn> </omvsn>
       <xmlvsn> </xmlvsn>
       <statime> </statime>
       <stotime> </stotime>
       <staseq> </staseq>
       <stoseq> </stoseq>
       <rqsttkn1> </rqsttkn1>
       <rc> </rc>
       <rsn> </rsn>
   </ctl>
   <cmderr>
     <mbr name="membername">
       <typ> </typ>
       <styp> </styp>
       <rc> </rc>
       <rsn> </rsn>
     </mbr>
   </cmderr>
   <cmdsecerr>
     <exit>
        <rc> </rc>
        <userdata> </userdata>
     </exit>
     <saf>
        <rc> </rc>
        <racfrc> </racfrc>
        <racfrsn> </racfrsn>
     </saf>
   </cmdsecerr>
   <cmd>
     <master> </master>
     <userid> </userid>
     <verb> </verb>
     <kwd> </kwd>
     <input> </input>
   </cmd>
   <cmdrsphdr>
     <hdr ... /hdr>
   </cmdrsphdr>
   <cmdrspdata>
```

```
      <rsp> </rsp>
   </cmdrspdata>
   <msgdata>
     <mbr name="membername">
       <msg> </msg>
     </mbr>
   </msgdata>
</imsout>
```

CSLOMQRY output

Each of the command syntax examples contain a sample of CSLOMQRY XML output. The examples present different scenarios that generate XML output based on the commands that are used in the example.

CSLOMQRY XML output

The tags in the following output example can be returned as a result of a CSLOMQRY request.

The command syntax and translatable text that is returned as a result of the CSLOMQRY QUERY TYPE(CMDSYNTAX) request includes information for type-2 commands.

```
<imsout>
   <ctl>
       <omname> </omname>
       <omvsn> </omvsn>
       <xmlvsn> </xmlvsn>
       <statime> </statime>
       <stotime> </stotime>
       <staseq> </staseq>
       <stoseq> </stoseq>
       <rqsttkn1> </rqsttkn1>
       <rc> </rc>
       <rsn> </rsn>
   </ctl>
   <cmdclients>
     <mbr name="membername">
       <typ> </typ>
       <styp> </styp>
       <vsn> </vsn>
       <jobname> </jobname>
     </mbr>
   </cmdclients>
   <cmdsyntax> </cmdsyntax><cmddtd>
       <!ELEMENT imsout ( ctl, cmdclients?,   cmdsyntax?, cmddtd?,
       cmdtext?, cmderr?, cmd, cmdrsphdr, cmdrspdata?, msgdata? )>
       <!ELEMENT ctl (omname?, omvsn?, xmlvsn?,   stattime, stotime,
       statseq, stoseq,   rqsttkn1?, rqsttkn 2?, rc, rsn )>
       <!ELEMENT omname (#PCDATA) >
       <!ELEMENT omvsn (#PCDATA) >
       <!ELEMENT xmlvsn (#PCDATA) >
       <!ELEMENT statime (#PCDATA) >
       <!ELEMENT stotime (#PCDATA) >
       <!ELEMENT staseq (#PCDATA) >
       <!ELEMENT stoseq (#PCDATA) >
       <!ELEMENT rqsttkn1 (#PCDATA) >
       <!ELEMENT rqsttkn2 (#PCDATA) >
       <!ELEMENT rc (#PCDATA) >
       <!ELEMENT rsn (#PCDATA) >
       <!ELEMENT cmdclients ( mbr+ )>
       <!ELEMENT cmdsyntax (#PCDATA) >
       <!ELEMENT cmddtd (#PCDATA ) >
       <!ELEMENT cmdtext (#PCDATA) >
       <!ELEMENT cmderr ( mbr* )>
       <!ELEMENT MBR ( (TYP, STYP, ((VSN, JOBNAME) | (rc, rsn))) | msg)>
```

```
       <!ELEMENT typ (#PCDATA) >
       <!ELEMENT styp (#PCDATA) >
       <!ELEMENT vsn (#PCDATA) >
       <!ELEMENT jobname (#PCDATA) >
       <!ELEMENT msg (#PCDATA) >
       <!ELEMENT cmdsecerr ( exit, saf )>
       <!ELEMENT exit ( rc, userdata ) >
       <!ELEMENT saf ( rc, racfc, racfrsn )>
       <!ELEMENT userdata (#PCDATA) >
```

```
        <!ELEMENT racfc (#PCDATA) >
        <!ELEMENT racfrsn (#PCDATA) >
        <!ELEMENT cmd ( master?, userid?, verb, kwd, input )>
        <!ELEMENT master (#PCDATA) >
        <!ELEMENT userid (#PCDATA) >
        <!ELEMENT verb (#PCDATA) >
        <!ELEMENT kwd (#PCDATA) >
        <!ELEMENT input (#PCDATA) >
        <!ELEMENT cmdrsphdr ( hdr* ) >
        <!ELEMENT hdr (#PCDATA) >
        <!ELEMENT cmdrspdata ( rsp* ) >
        <!ELEMENT rsp (#PCDATA) >
        <!ELEMENT msgdata ( mbr ) >
        <!ATTLIST hdr slbl   CDATA #REQUIRED >
        <!ATTLIST hdr llbl   CDATA #REQUIRED >
        <!ATTLIST hdr scope  CDATA #REQUIRED >
        <!ATTLIST hdr sort   CDATA #REQUIRED >
        <!ATTLIST hdr key    CDATA #REQUIRED >
        <!ATTLIST hdr scroll CDATA #REQUIRED >
        <!ATTLIST hdr len    CDATA #REQUIRED >
        <!ATTLIST hdr dtype  CDATA #REQUIRED >
        <!ATTLIST hdr align  CDATA #REQUIRED >
</cmddtd>
<cmdtext> </cmdtext>
</imsout>
```

Related reference

“CSLOMQRY: query request” on page 121
With the CSLOMQRY request, any AOP client that is running on the host can request OM-specific information.

CSLOMOUT output

The command syntax example contains a sample of CSLOMOUT XML output. The example presents a scenario that generates XML output based on the commands that are used in the example.

Unsolicited output message encapsulated in XML tags

The tags in the following output example can be returned as a result of a CSLOMOUT request.

```
<imsout>
  <ctl>
      <omname></omname>
      <omvsn></omvsn>
      <xmlvsn></xmlvsn>
      <statime></statime>
      <staseq></staseq>
      <uom>UOM</uom>
   </ctl>
   <msgdata>
      <mbr name="membername">
         <typ></typ>
         <styp></styp>
         <msg></msg>
      </mbr>
   <msgdata>
</imsout>
```

XML tags returned as CSL OM responses

Different XML tags can be returned as CSL OM responses. Each tag name is delimited by the characters < and >. Tags can be nested within parent tags to encapsulate related information.

Data or other sets of tags are contained between these start and end tags. In the list of tags, indentation indicates that the tags are nested within the parent tags.

<?xml version "1.0"?>
The version of XML used in this output.

<!DOCTYPE imsout SYSTEM "imsout.dtd">
The DOCTYPE tag identifies the file that contains the document type definition (DTD). The DTD describes the structure that is supported for this type of XML document. Users of z/OS can find the DTD information in the CSLOMDTD member, located in the IMS.SDFSRESL data set.

<imsout> </imsout>
The <imsout> </imsout> tags encapsulate the output from OM. These tags are returned on every request.

<ctl> </ctl>
The <ctl> </ctl> tags encapsulate the control information that is returned by OM. These tags are returned on every request and include the following control information:

<omname>*om name*</omname>
Indicates the name of the OM that processed this request. The name is specified on the OMNAME= execution parameter of the CSLOIxxx PROCLIB member.

<omvsn>*om version number*</omvsn>
Indicates the OM version number.

<xmlvsn>*xml version number*</xmlvsn>
Indicates the XML version number.

<statime>*starttime*</statime>
Indicates the time that OM started processing the request. The field is in the following format: yyyy.ddd hh:mm:ss.th

<stotime>*stoptime*</stotime>
Indicates the time that OM completed request processing. The field is in the following format: yyyy.ddd hh:mm:ss.th

<staseq>*startsequence*</staseq>
Indicates the sequence value when OM started processing the request. This value can be used for sorting and is in printable EBCDIC hexadecimal format.

<stoseq>*stopsequence*</stoseq>
Indicates the sequence value when OM stopped processing the request. This value can be used for sorting and is in printable EBCDIC hexadecimal format.

<rqsttkn1>*requesttoken1*</rqsttkn1>
Indicates the user-specified RQSTTKN1 value that is associated with the response. OM converts unprintable characters to periods (.) in the output.

<rqsttkn2>*requesttoken2*</rqsttkn2>
Indicates the user-specified RQSTTKN2 value that is associated with the response. OM converts unprintable characters to periods (.) in the output.

<rc>*returncode*</rc>
The return code for the request in printable EBCDIC hexadecimal format.

<rsn>*reasoncode*</rsn>
The reason code for the request in printable EBCDIC hexadecimal format.

<uom>*unsolicited output message*</uom>
Indicates that the XML is for an unsolicited output message

<cmdclients> </cmdclients>
Encapsulates information about OM clients. These tags can be returned on a QUERY(CMDCLIENTS) request.

<mbr name="*membername*"></mbr>
Indicates the name of the IMSplex member that is registered for commands.

<typ>*membertype*</typ>
Indicates the type of IMSplex member.

<styp>*membersubtype*</styp>
Indicates the IMSplex member subtype. OM converts unprintable characters to periods (.) in the output.

<vsn>*memberversion*</vsn>
Indicates the member version number.

<jobname>*memberjobname*</jobname>
Indicates the member job name.

<cmddtd> </cmddtd>
Encapsulates the Document Type Definition (DTD) that are defined by OM for command syntax and OM output XML. These tags can be returned on a QUERY(CMDSYNTAX) request.

<cmdsyntax> </cmdsyntax>
Encapsulates the XML definitions for the commands that are registered to OM from all of its clients. These tags can be returned on a QUERY(CMDSYNTAX) request.

<cmdtext> </cmdtext>
Encapsulates the translatable text strings that are associated with the XML command syntax tags. These tags can be returned on a QUERY(CMDSYNTAX) request.

<cmd> </cmd>
Encapsulates the command information that was passed to OM. These tags can be returned on a command request. The output returned in these tags is what was provided on the CMD= parameter on the CSLOMBLD macro. The following tags are included within the <cmd> tags:

<master> </master>
Encapsulates the name of the command processing client that was tagged as the master when sending the command. This information will not be present unless the command was successfully sent to at least one command processing client.

<userid> </userid>
Encapsulates the user ID of the originator of the command.

<verb> </verb>
Encapsulates the short form of the command verb that was processed by OM. The verb might have been passed to OM in a long form.

<kwd> </kwd>
Encapsulates the command keyword that was processed by OM.

<input> </input>
Encapsulates the actual input command string that was passed to OM. The following characters are converted to periods (.) to maintain the validity of the output XML:

- Greater than signs (>)
- Less than signs (<)
- Ampersands (&)
- Non-printable characters

<cmdrsphdr> </cmdrsphdr>
Encapsulates the command header information that describes the data fields returned in the command response. These tags can be returned on a command request.

<hdr ... />
Defines the attributes of columns of data fields.

The command response header information is in the format shown in the following format example:

```
<hdr slbl="ss" llbl="llll" scope="c" sort="d" key="e" scroll="f" len="g" dtype="h" align="i"
skipb="no"/>
```

slbl
Short label used to match data description with data value that is returned by the <cmdrspdata> tag.

The short label values vary by command. Refer to the documentation for each command to determine what values can be returned for a specific command.

llbl
Long label that can be used as the table column header.

The long label values vary by command. Refer to the documentation for each command to determine what values can be returned for a specific command.

scope
Indicates if the data is global or local.

GBL
Indicates that the data is global. For query output, global data applies to all resources of the same name, but is returned only once in the command response for a specific resource name. Global information applies to other rows of the same resource name for different IMSplex member names. The resource name is the data field identified by a KEY="1" attribute. If an application chooses to transform the command response data into a table to be displayed for a user, the global data value can be propagated to other rows for the same resource name.

LCL
Indicates that the data is local. For query output, local data applies only to a specific resource name in a specific IMS. Different IMS systems can return different values for local data fields. Each IMS returns its local value when it is available. If an application chooses to transform the command response data into a table to be displayed for a user, the local data value should not be propagated to other rows for the same resource name.

sort
Indicates whether or not this field should be sorted or the sort direction.

A
Sort in ascending order.

D
Sort in descending order.

N
Do not sort field.

key
Indicates the sort priority for this field.

0
Field is not sorted.

1
The highest priority sort field.

2
The second highest priority sort field.

n
The *n* the priority sort field.

The priority value indicated on KEY= in the <cmdrsphdr> tag has been predetermined. Some command responses can specify multiple sort fields. Several fields are listed within the <cmdrsphdr> tags with their sort priorities:

- Trancode - 1
- MbrName - 4
- CC - 0
- PSBname - 0
- QCnt - 2
- LCls - 0
- LQCnt - 3

The following code example causes the command results to be sorted with the following priority:

1. Trancode
2. Qcnt
3. LQcnt
4. MbrName

```
<?xml version="1.0"?>
<!DOCTYPE imsout SYSTEM "imsout.dtd">
<imsout>
<ctl>
<omname>OM1OM   </omname>
<omvsn>1.1.0</omvsn>
<xmlvsn>1   </xmlvsn>
<statime>2002.261 18:33:56.425140</statime>
<stotime>2002.261 18:33:56.487941</stotime>
<staseq>B8400987409B4A0E</staseq>
<stoseq>B84009874FF05409</stoseq>
<rqsttkn1>USRT002 10113356</rqsttkn1>
<rc>00000000</rc>
<rsn>00000000</rsn>
</ctl>
<cmd>
<master>IMS2    </master>
<userid>USRT002 </userid>
<verb>QRY </verb>
<kwd>TRAN           </kwd>
<input>QRY TRAN NAME(ADD*) SHOW(PSB,QCNT,CLASS) </input>
</cmd>
<cmdrsphdr>
<hdr slbl="TRAN" llbl="Trancode" scope="LCL" sort="a" key="1"
scroll="no" len="8" dtype="CHAR" align="left" />
<hdr slbl="MBR" llbl="MbrName" scope="LCL" sort="a" key="4"
scroll="no" len="8" dtype="CHAR" align="left" />
<hdr slbl="CC" llbl="CC" scope="LCL" sort="n" key="0"
scroll="yes" len="4" dtype="INT" align="right" />
<hdr slbl="PSB" llbl="PSBname" scope="LCL" sort="n" key="0"
scroll="yes" len="8" dtype="CHAR" align="left" />
<hdr slbl="Q" llbl="QCnt" scope="GBL" sort="d" key="2"
scroll="yes" len="8" dtype="INT" align="right" />
<hdr slbl="LCLS" llbl="LCls" scope="LCL" sort="n" key="0"
scroll="yes" len="3" dtype="INT" align="right" />
<hdr slbl="LQ" llbl="LQCnt" scope="LCL" sort="d" key="3"
scroll="yes" len="8" dtype="INT" align="right" /></cmdrsphdr>
<cmdrspdata>
<rsp>TRAN(ADDPART ) MBR(IMS2    ) CC(   0) PSB(DFSSAM04) LCLS(  4)
LQ(   0) </rsp>
<rsp>TRAN(ADDINV  ) MBR(IMS2    ) CC(   0) PSB(DFSSAM04) LCLS(  4)
LQ(   0) </rsp>
<rsp>TRAN(ADDPART ) MBR(IMS2    ) CC(   0) Q(    0) </rsp>
<rsp>TRAN(ADDINV  ) MBR(IMS2    ) CC(   0) Q(    0) </rsp>
<rsp>TRAN(ADDPART ) MBR(SYS3    ) CC(   0) PSB(DFSSAM04) LCLS(  4)
LQ(   0) </rsp>
<rsp>TRAN(ADDINV  ) MBR(SYS3    ) CC(   0) PSB(DFSSAM04) LCLS(  4)
LQ(   3) </rsp>
</cmdrspdata>
</imsout>
```

If two records have the same Trancode, they are sorted by Qcnt. If they also have the same Qcnt, they are sorted by LQcnt. If they have the same LQcnt value, they are sorted by MbrName, and so on, until the n^{th} sort field is used.

The results of the XML in the previous code are displayed in the following code sample.

```
Response for: QRY TRAN NAME(ADD*) SHOW(PSB,QCNT,CLASS)

Trancode MbrName    CC PSBname      QCnt LCls    LQCnt
ADDINV   IMS2        0                 0
ADDINV   SYS3        0 DFSSAM04             4        3
ADDINV   IMS2        0 DFSSAM04             4        0
ADDPART  IMS2        0                 0
ADDPART  IMS2        0 DFSSAM04             4        0
ADDPART  SYS3        0 DFSSAM04             4        0
```

Depending on which fields were selected using the SHOW parameter of the **QUERY** command, not all intermediate priority value fields will be displayed. That is, the results could display fields whose priority values were set at 1 and 4, but not display fields whose priority values were set at 2 and 3. A program might leave the records in the original order, sort them using the predetermined priority values, or sort by other fields using criteria set locally by the user.

scroll
Indicates whether this field is scrolled off of the screen when TSO SPOC shifts the screen to the right.

NO
Do not scroll the field.

YES
Allow the field to scroll off the screen.

len
Maximum length of data (data returned could contain fewer characters). If a table of data is being created from the output response, this value can be used to determine the width of the column that is displayed for this attribute. If the value for this field is '*', this is a variable length field.

dtype
Describes the original data type. All data is returned in character format. However, some fields represent numeric data. Data that originated as integer might need to be converted from character to integer in order to perform mathematical calculations.

CHAR
The output field represents character data.

INT
The output field is the character representation of integer data.

align
Indicates recommended column alignment if data is to be formatted into columns.

RIGHT
Data is right-aligned, for example, numeric data.

CENTER
Data is centered.

LEFT
Data is left-aligned, for example, character data.

skipb

no
The column is displayed on the TSO SPOC output, even if no client returned any information for this column. This is the default.

yes
The column is not displayed on the TSO SPOC output if no client returned any information for this column.

<cmdrspdata> </cmdrspdata>
Encapsulates the command response detail information. These tags can be returned on a command request. The tags contain the actual data that is described by the <cmdrsphdr> </cmdrsphdr> tags.

Refer to the documentation for each command to determine what values can be returned for a specific command.

<rsp>*response data*</rsp>

Contains a logical line of command response output for a particular resource. The response data contains various tags in the form *name(value)*. The name maps to short label (slbl=) values in the <hdr> tag. This is shown in the following example, with the values TRAN and PSB.

```
<cmdrsphdr>
<hdr slbl="TRAN"    llbl="Trancode"... />
<hdr slbl="PSB"     llbl="PSBname" ... />
</cmdrsphdr>
<cmdrspdata>
<rsp>TRAN(A        ) PSB(A11      ) </rsp>
<rsp>TRAN(B        ) PSB(B22      ) </rsp>
<rsp>TRAN(C        ) PSB(C33      ) </rsp>
</cmdrspdata>
```

The <hdr> tag includes a long label value (llbl=), which can be used as column headings. This is shown in the following output example, specifically Trancode and PSBname.

```
Trancode  PSBname
A         A11
B         B22
C         C33
```

The values included in the response data propagate the data columns of the SPOC output. Other tags in the <hdr> tag describe formatting attributes for values in that column.

<msgdata> </msgdata>

Encapsulates pre-built IMS messages. The messages can be of any type including informational, warning, or error messages. These tags can be returned on a command request.

<mbr name="*membername*"></mbr>

Indicates the name of the IMSplex member that returned the message.

<msg>*message data*</msg>

Contains a logical command response output for a resource in a message format. The message starts with a message number (for example, DFS*nnnn*I). There is no LL field or X'15' new line character.

<cmderr> </cmderr>

Encapsulates the return and reason code information returned by OM or a command processing client. These tags are returned on command requests when an error specific to a command processing client must be returned. For each IMSplex member with an error, the following information is returned.

<mbr name="*membername*"></mbr>

Indicates the name of the IMSplex member for which an error was detected.

<typ>*membertype*</typ>

Indicates the type of IMSplex member.

<styp>*membersubtype*</styp>

Indicates the IMSplex member subtype. OM converts unprintable characters to periods (.) in the output.

<rc>*returncode*</rc>

Indicates the return code for the IMSplex member in printable EBCDIC hexadecimal format.

<rsn>*reasoncode*</rsn>

Indicates the reason code for the IMSplex member in printable EBCDIC hexadecimal format.

<cmdsecerr> </cmdsecerr>

Encapsulates the return and reason code information returned by the OM security exit, SAF and RACF, or equivalent. If the OM security exit rejected the command for any reason, the user data from the security exit is also encapsulated here.

<exit> </exit>

Encapsulates the return code and user data from the OM security exit.

<rc>*returncode*</rc>
Indicates the return code from the OM security exit in printable EBCDIC hexadecimal format.

<userdata>*userdata*</userdata>
Indicates the user data returned from the OM security exit in the OSCX_USERDATA field of the OM Command Security user exit parameter list (CSLOSCX). OM converts unprintable characters to periods (.) in the output.

<saf> </saf>
Encapsulates the return and reason codes from the SAF and RACF or an equivalent.

<rc>*returncode*</rc>
Indicates the return code from the SAF in printable EBCDIC hexadecimal format.

<racfrc>*racfreturncode*</racfrc>
Indicates the return code from RACF or equivalent security product in printable EBCDIC hexadecimal format.

<racfrsn>*racfreasoncode*</racfrsn>
Indicates the reason code from RACF or equivalent security product in printable EBCDIC hexadecimal format.

Related reference

“CSLOMI XML output examples” on page 233
Each of the command syntax examples contain a sample of CSLOMI XML output. The examples present different scenarios that generate XML output based on the commands that are used in the example.

Chapter 10. REXX SPOC API and the CSL

The REXX SPOC API allows REXX programs to submit commands to OM and to retrieve the command responses.

REXX SPOC API environment with the CSL OM

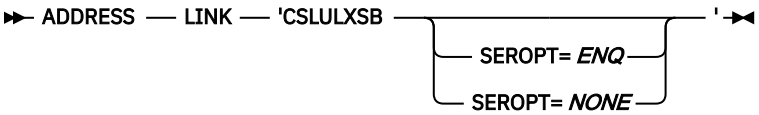
The REXX SPOC API allows REXX programs to set up the IMSplex environment, submit commands to OM, and to retrieve the command responses.

There are three phases related to using the REXX SPOC API with the CSL OM:

1. Set up the REXX environment
2. Set up the IMSplex environment and issue commands
3. Retrieve the command responses

Setting up the REXX environment in a CSL

By issuing the **ADDRESS** command, you can call the program CSLULXSB to set up the REXX environment. This program establishes the REXX subcommand environment for the REXX SPOC API.



Note: Other forms of the **ADDRESS** command might not work in the Tivoli® NetView® for z/OS environment.

Keyword of the LINK statement

The LINK statement supports the following keyword:

SEROPT=ENQ|NONE
Specifies whether a serialization enqueue (ENQ) needs to be obtained to serialize parallel instances of the IMS REXX SPOC command environment that are started under the same address space. If this keyword is specified in the LINK statement, it overrides the default or the system level specification, which is specified in the CSLULXD0 user exit. If this keyword is not specified, the default or system level specification in the CSLULXD0 user exit is used.

Important: Each REXX SPOC instance that is in the same address space must use the same serialization option. When serialization is required,**SEROPT=**ENQ (either in the LINK CSLULXSB command, or in the CSLULXD0 exit) must be specified by every REXX SPOC instance in the same address space.

You can specify one of the following values for this keyword:

ENQ
SEROPT=ENQ indicates that multiple IMS REXX SPOC instances might be started under the same address space and a serialization ENQ is required to serialize the subcommand environment.

The ENQ option specifies the STEP option so that the scope is limited to the address space. The ENQ resource qname is *CSLUSPCA*, and the rname is *CSLULXTP||jobid*. You can use the **D GRS,RES=(CSLUSPCA,*)** command to display serialization information.

When you set up an IMS REXX subcommand environment, a step level ENQ for resource `CSLULXTP||jobid` is obtained. This ENQ will not be released until the IMS REXX **END** subcommand is processed.

Important: Ensure that either the REXX SPOC application always issues an IMS REXX **END** subcommand to dequeue instances, or that the started task ends in a timely manner, if the IMS

REXX subcommand environment is started to prevent subsequent instances in the same address space from waiting on the ENQ.

NONE
SEROPT=NONE indicates that multiple IMS REXX SPOC instances are not started under the same address space and a ENQ serialization is not required.

The following examples show how to enable or disable the serialization by specifying the **SEROPT** parameter:

Enabling serialization for parallel REXX applications with SEROPT=ENQ

```
/*-------------------------------------------------------------------
| When multiple IMS Rexx SPOC instances could be started           |
| under the same address space, then an ENQ is needed to           |
| serialize the subcommand environment.                            |
| The Rexx SPOC program requests this by specifying                |
| SEROPT=ENQ on the Address LINK 'CSLULXSB' statement.             |
-------------------------------------------------------------------*/
Address LINK 'CSLULXSB SEROPT=ENQ'
if rc = 0 then
  do
    Address IMSSPOC

    "IMS  IPLX4"
    "ROUTE IMS1,IMSB"

    "WAIT  5:00"

    "CART DISTRAN"
    "/DIS TRAN PART"

     spoc_rc = cslulgts('DISINFO.','DISTRAN',"59")
     do z1 = 1 to  DISINFO.0
       /* display each line of XML information */
       Say disinfo.z1
     end
     "END"
  End
```

Disabling serialization for parallel REXX applications with SEROPT=NONE

```
/*-------------------------------------------------------------------
| When only a single IMS Rexx SPOC instance is started             |
| under an address space, then an ENQ is not needed to             |
| serialize the subcommand environment.                            |
| The Rexx SPOC program requests this by specifying                |
| SEROPT=NONE on the Address LINK 'CSLULXSB' statement.            |
| This is the default (SEROPT can be omitted).                     |
-------------------------------------------------------------------*/
Address LINK 'CSLULXSB SEROPT=NONE'
if rc = 0 then
  do
    Address IMSSPOC

    "IMS  IPLX4"
    "ROUTE IMS1,IMSB"

    "WAIT  5:00"

    "CART DISTRAN"
    "/DIS TRAN PART"

     spoc_rc = cslulgts('DISINFO.','DISTRAN',"59")
     do z1 = 1 to  DISINFO.0
       /* display each line of XML information */
       Say disinfo.z1
     end
     "END"
  End
```

Related reference
CSLULXD0 user exit (Exit Routines)

Setting up the IMSplex environment

By switching the default host command to IMSSPOC and issuing the **ADDRESS** command, you can set up the IMSplex environment. After you set the default host command to IMSSPOC, IMSSPOC executes subsequent host commands issued by the REXX program that is running.

After you set the default host command to IMSSPOC, IMSSPOC executes subsequent host commands issued by the REXX program that is running. You can switch to other host commands by using the **ADDRESS** command with other hosts. For example:

```
ADDRESS TSO
ADDRESS MVS
ADDRESS ISPEXEC
```

You can then issue commands that are specific to those environments.

Note: If you issue commands other than the subcommands described here in the REXX environment, they are sent to OM for processing.

IMS subcommand

The IMS subcommand establishes the name of the IMSplex. You must issue the IMS subcommand to establish the IMSplex name before any other commands can be issued. A prefix of "CSL" is automatically added to the name that you specify.

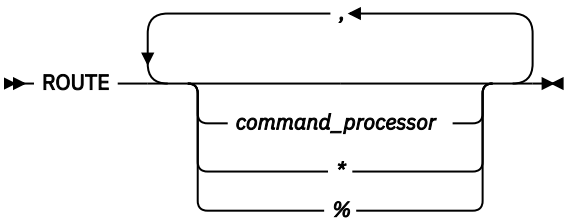
ROUTE subcommand

Use the ROUTE subcommand to set the name of the command processors. The command processors are the specific systems that will execute subsequent IMS commands. If you do not specify a command processor:

- the previous routing value is removed
- commands are routed to all members of the IMSplex. This is the default.

If "*" is specified, the command is routed to all registered command processing clients in the IMSplex. If "%" is specified, the command is routed to only one command processing client in the IMSplex that is registered for the command and that has MASTER capability. The Operations Manager chooses the command processing client.

The ROUTE subcommand is optional.



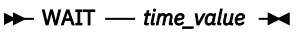
CART subcommand

Use the Command and Response Token (CART) subcommand to set the name of the command and response token. This 16-character text string token is used to retrieve the command response.

You must issue the CART subcommand before you can issue any IMS commands.

WAIT subcommand

Use the WAIT subcommand to provide a timeout value to OM. The time value must be in the form MMM:SS or ssss. The maximum value you can specify is 999:59. The WAIT subcommand is optional.



Issuing type-2 IMS commands

You issue IMS commands, including type-2 commands, by including them in the REXX program stream as quoted strings or as REXX variables that resolve to quoted strings.

Examples of type-2 commands

```
"QUERY IMSPLEX SHOW(ALL)"
```

```
"DIS ACT"
```

```
tranlist = "PETER1,MATT1"
"QUERY TRAN NAME("tranlist")"
```

CSLULGTS: retrieving command responses in XML

By issuing the CSLULGTS request, you can retrieve command responses. The CSLULGTS command puts the command responses to a stem variable so that REXX can access them.

stem_name
After the CSLULGTS command completes successfully, the stem variable contains XML statements. Each row of the stem variable contains one XML statement. If the beginning and ending XML tags are adjacent (that is, no other XML tags exist between them), they are placed in the same row of the stem variable. A single row of a stem variable might look like this:

```
<rsp>TRAN(VIDB    ) MBR (IMS2    ) CC(    0) </rsp>
```

token_name
The name of the command and response token (CART). It should match the name specified on the CART subcommand.

wait_time
A timeout value for the CSLULGTS command. The CSLULGTS command waits until the command completes, but the wait lasts only as long as the time specified. The wait time is in the format MMM:SS or ssss. The maximum timeout value is 999:59. Enclose this value in quotes.

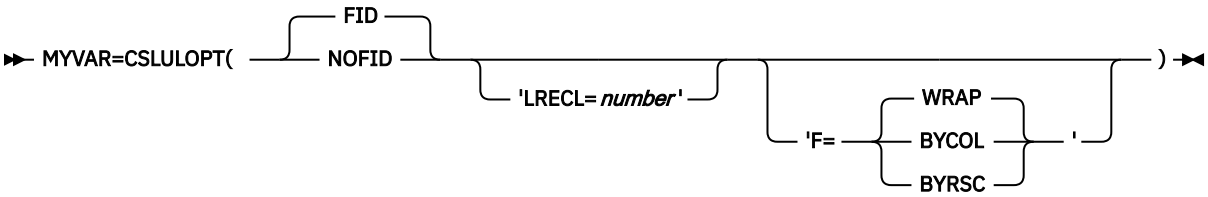
Note: This timeout value is not the same as the timeout value for the WAIT subcommand; however, this *wait_time* should be at least as long as the value specified on the WAIT subcommand. Otherwise, no command response are received for long running commands.

If no response is received the first time, you can issue the CSLULGTS command again.

CSLULOPT: including format identifiers in command responses

By issuing the CSLULOPT request, you can specify whether a command response should contain format IDs. Automated operator programs (AOPs) use format identifiers (FID) to identify the record format of specific lines of a command response.

Invoke CSLULOPT before you issue the IMS operator command. The setting you select on CSLULOPT is in effect for this REXX program until you explicitly change it.



F

Specifies an output option. The possible options are:

BYCOL
Group lines by column.

BYRSC
Group lines by resource.

WRAP
Wrap individual lines (default).

FID

Specifies that the command response includes the FID. The default is FID.

LRECL

Specifies the logical record length as a numeric value.

NOFID

Specifies that the command response does not include the FID.

MYVAR

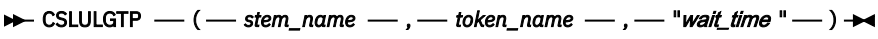
A variable that you can specify that contains the return code.

Related reference

REXX SPOC return and reason codes (Messages and Codes)

CSLULGTP: retrieving command responses directly to a REXX stem variable

By issuing the CSLULGTP request, you can retrieve command responses from IMS Operations Manager (OM) and put the command response into a REXX stem variable. The REXX program then refers to the information in the stem variable, rather than parsing XML statements as it does with the CSLULGTS request.



stem_name

After the CSLULGTP request completes, the REXX stem variable is populated with the command response that is returned by OM. The REXX program can then refer to the command response for further operations.

During initialization, the REXX stem variable is set to null before being processed. Any fields that are not explicitly set as output by the CSLULGTP request will be null on return from the CSLULGTP call.

token_name

The name of the command and response token (CART). The token name should match the name specified on the CART subcommand.

wait_time

A timeout value for the CSLULGTP command. The CSLULGTP command waits until the command completes, but the wait lasts only as long as the time specified. The wait time is in the format MMMMM:SS or ssssss. Enclose this value in quotes.

The maximum timeout value is 99999:59. If you do not specify a value for this parameter, the command times out after a very short delay of less than one tenth of a second.

IMS checks whether the command completed or timed out every 0.01 seconds if the value of **wait_time** is less than ten seconds. If the value is larger than ten seconds, IMS checks every second instead.

This timeout value is not the same as the timeout value for the **WAIT** subcommand; however, the value of **wait_time** should be at least as long as the value specified on the **WAIT** subcommand. Otherwise, no command response are received for long running commands.

If no response is received the first time, you can issue the CSLULGTP command again.

Intermittent results can occur when **stem_name**, **token_name**, and **wait_time** parameters are not coded on the CSLULGTP call.

Sample code for retrieving command responses using the CSLULGTP request

The following examples provide a code sample of the CSLULGTP request.

Example #1: In the example, the IMS command **QRY TRAN NAME(A)** is issued, and the CSLULGTP request is used to retrieve the command response.

```
Address LINK  'CSLULXSB'
Address IMSSPOC
"ims PLEX1"
"wait 5:00"
cartid = 'PROD12'
"CART" cartid
"QRY TRAN NAME(A*)"
results = cslulgtp('qinfo.', cartid,"5:00")
If qinfo.ctl.rc = 0 Then
  Do
    say "OM name        =("qinfo.ctl.omname")"
    say "command master =("qinfo.cmd.master")"
  End
```

Example #2: This program issues the **IMS UPD PGM** command and processes the command responses. The following actions are taken based on the completion code:

```
  0  - OK - say command complete for program 'program name'
  10 - not found - issue CRE PGM command
  73 - PSB scheduled - issue /DIS ACT REG command
  other completion code - say invalid CC for program

/*-------------------------------------------------------------
| Establish IMS rexx environment.                             |
-------------------------------------------------------------*/
Address LINK 'CSLULXSB'
Address IMSSPOC
"IMS   PLEX1"
"WAIT  5:00"
"CART  CMDTOKEN"

/*-------------------------------------------------------------------------
| Issue an IMS command                                                    |
-------------------------------------------------------------------------*/
   "UPD PGM NAME(APOL1,BMP255,PGMX,PGMY) SET(SCHDTYPE(SERIAL))"

/*-------------------------------------------------------------------------
| Get command responses                                                   |
-------------------------------------------------------------------------*/
spoc_rc = CSLULGTP('stem1.','CMDTOKEN',"59")

/*-------------------------------------------------------------------------
| Get some data for diag                                                  |
-------------------------------------------------------------------------*/
Say '*DATA: '
Say '*CMD issued = '               stem1.cmd.input
Say '*SPOC rc    = '               spoc_rc
Say '*Command rc = '               stem1.ctl.rc
Say '*Command rsn= '               stem1.ctl.rsn
row = 1
if stem1.cmderr.0 > 0 then do
  Say '*Num of errors    ='        stem1.cmderr.0
  Say '*CMD error RC     ='        stem1.cmderr.row.rc
  Say '*CMD error RSN    ='        stem1.cmderr.row.rsn
end /* if */
```

```
if stem1.rsp.0    > 0 then do
    Say '*Num of rsp rows ='             stem1.rsp.0
end /* if */
if stem1.rsp.0    > 0 then do
 /* print each line of the stem.report  */
 Do n = 1 to stem1.report.0
  Say stem1.report.n
 End
end /* if */
say ' '

/*--------------------------------------------------------------------------
| Process responses                                                        |
--------------------------------------------------------------------------*/
spoc_rc = left(spoc_rc,2)
if spoc_rc >< '08' then do              /* process response if GTP ran ok */

   /*-------------------------------------------------------------
   | Process the stem response if IMS return code is             |
   | X'00000000' or X'00000004' or X'0000000C'.                  |
    -------------------------------------------------------------*/
   row = 1
   imsrc = left(stem1.cmderr.row.rc,2)
   If (imsrc = 00) | (imsrc = 04) | (ims.rc = 0C) then do

    if stem1.RSP.0 > 0 then do
     cccol = 3      /* set col num for CC */
     cctxtcol = 4   /* set col num for CCtext */

     do  row = 1 to stem1.RSP.0
      stem_cc.row = stem1.RSP.row.cccol
      stem_cctxt.row = stem1.RSP.row.cctxtcol
      say '*Completion code on row 'row ': 'stem_cc.row
      say '*Completion code text on row 'row ': 'stem_cctxt.row
      stem_cc.row = stem1.RSP.row.cccol
      stem_cctxt.row = stem1.RSP.row.cctxtcol
      select
        /*-------------------------------------------------------------
        | Process case 1                                              |
         -------------------------------------------------------------*/
        when left(stem_cc.row,2) = 0  then do
           say 'Command complete for program' stem1.RSP.row.1
           say ' '
        end

        /*-------------------------------------------------------------
        | Process case 2.                                             |
        | When completion code is X'10', resource not found, issue    |
        | a CREATE PGM command to create a program like the default   |
        | descriptor.                                                 |
         -------------------------------------------------------------*/
        when left(stem_cc.row,2) = 10 then do
           say 'Issue CREATE PGM command for' stem1.RSP.row.1
           arg1 = stem1.rsp.row.1       /* extract pgmname to arg1 */
           "CREATE PGM NAME("arg1")"    /* issue CRE cmd with arg1 */
           spoc_rc1 = CSLULGTP('stem2.','CMDTOKEN',"59")
           if stem2.rsp.0    > 0 then do
             /* print each line of the stem.report  */
             Do n = 1 to stem2.report.0
               Say stem2.report.n
             End
             say ' '
           end /* if */
        end /* when */

        /*-------------------------------------------------------------
        | Process case 3                                              |
        | When completion code is X'73', program scheduled, issue     |
        | a DISPLAY ACTIVE REGION command to see the active regions.  |
        | The region needs to be stopped and the UPD command retried. |
         -------------------------------------------------------------*/
        when left(stem_cc.row,2) = 73 then do
           say 'Program ' stem1.RSP.row.1 'is scheduled'
           "/DISPLAY ACT REGION"
           spoc_rc2 = CSLULGTP('stem3.','CMDTOKEN',"59")
           /* print the response from each IMS */
           if stem3.MSGDATA.MSG.1.0 > 0 then do
             do x = 1 TO stem3.MSGDATA.NAME.0
               SAY stem3.MSGDATA.NAME.X
               do y = 1 TO stem3.MSGDATA.MSG.X.0
                 SAY stem3.MSGDATA.MSG.X.Y
               end/*end do y loop */
```

```
              end /* end do x loop */
            end /* end if */
            say 'Issue /STOP REGION commnd and retry UPD cmd'
            say ' '
         end /* when */

         /*-------------------------------------------------------------
         | For all other completion codes print the error compcode.    |
          -------------------------------------------------------------*/
         otherwise do
            say 'Invalid CC for program' stem1.RSP.row.1
            say ' '
         end /* otherwise */

       end /* select */
      end  /* do loop */

   end /* if (imsrc = 00) | . . .*/

   /*-------------------------------------------------------------
   | Print IMS rc and rsn for all other error rc/rsn.            |
    -------------------------------------------------------------*/
   Else do
      say '*IMS RC & RSN = ' stem1.ctl.rc  stem1.ctl.rsn
   end /* else */

end /* if spoc_rc >< '08' */

/*-------------------------------------------------------------
| Exit program                                                 |
 -------------------------------------------------------------*/
"END"   /* SPOC */

EXIT   /* REXX */
```

The **say** instructions in the previous example refer to elements of the REXX stem variable. The CSLULGTP request sets the suffix of the stem variable. The following table shows the possible suffix variable names that are set when the CSLULGTP request creates the stem variable.

Table 68. Suffix variable names set by the CSLULGTP command

XML tag	**Variable name**
<?xml version="1.0"?>	stem.xmlversion
<!DOCTYPE imsout SYSTEM "imsout.dtd">	stem.dtd
<imsout>	N/A
<ctl>	N/A
<omname> </omname>	stem.ctl.omname
<omvsn> </omvsn>	stem.ctl.omvsn
<xmlvsn> </xmlvsn>	stem.ctl.xmlvsn
<statime> </statime>	stem.ctl.statime
<stotime> </stotime>	stem.ctl.stotime
<rqsttkn1> </rqsttkn1>	stem.ctl.rqsttkn1
<rc> </rc>	stem.ctl.rc
<rsn> </rsn>	stem.ctl.rsn
<rsnmsg> </rsnmsg>	stem.ctl.rsnmsg

Table 68. Suffix variable names set by the CSLULGTP command (continued)

XML tag	Variable name
<rsntxt> </rsntxt>	stem.ctl.rsntxt
</ctl>	N/A
<cmderr>	stem.cmderr.0
<mbr name="membername">	stem.cmderr.x.name
<typ> </typ>	stem.cmderr.x.typ
<styp> </styp>	stem.cmderr.x.styp
<rc> </rc>	stem.cmderr.x.rc
<rsn> </rsn>	stem.cmderr.x.rsn
<rsntxt> </rsntxt>	stem.cmderr.x.rsntxt
</mbr>	N/A
</cmderr>	N/A
<cmdsecerr>	N/A
<exit>	N/A
<rc> </rc>	stem.cmdsecerr.exit.rc
<userdata> </userdata>	stem.cmdsecerr.exit.userdata
</exit>	N/A
<saf>	N/A
<rc> </rc>	stem.cmdsecerr.saf.r
<racfrc> </racfrc>	stem.cmdsecerr.saf.racfrc
<racfrsn> </racfrsn>	stem.cmdsecerr.saf.racfrsn
</saf>	N/A
</cmdsecerr>	N/A
<cmd>	N/A
<master> </master>	stem.cmd.master
<userid> </userid>	stem.cmd.userid
<verb> </verb>	stem.cmd.verb
<kwd> </kwd>	stem.cmd.kwd
<input> </input>	stem.cmd.input
</cmd>	N/A

Table 68. Suffix variable names set by the CSLULGTP command (continued)	
XML tag	**Variable name**
<cmdrsphdr>	N/A
<hdr></hdr>	stem.hdr.0 (number of columns) stem.hdr.x.slbl stem.hdr.x.llbl stem.hdr.x.scope stem.hdr.x.sort stem.hdr.x.key stem.hdr.x.scroll stem.hdr.x.len stem.hdr.x.dtype stem.hdr.x.align
</cmdrsphdr>	N/A
<cmdrspdata>	N/A
<rsp> </rsp>	stem.rsp.0 (number of rows) stem.rsp.x.0 (number of cols) stem.rsp.x.y stem.rsp.x.y stem.rsp.x.y stem.rsp.x.y stem.rsp.x.y stem.rsp.x.y stem.rsp.x.y stem.rsp.x.y stem.rsp.x.y
</cmdrspdata>	N/A
<msgdata>	N/A
<mbr name="membername">	stem.msgdata.name.0 (num of systems) stem.msgdata.name.y (1 member name)
<msg> </msg>	stem.msgdata.msg.y.0 (num of msgs /sys) stem.msgdata.msg.y.x (1 message line)
</mbr>	N/A
</msgdata>	N/A
</imsout>	N/A
N/A	stem.report.0 (number of lines) stem.report.x (1 line of report)

Where the suffix variables:

stem
user-defined stem name

x
row number of command response

y
column number of command response

The CSLULGTP function creates a report as part of the stem variable. The stem is any user provided value; the suffix is "report".

Important: When using Type-1 commands, use the tag `msgdata` instead.

```
"QRY TRAN SHOW(PSB,QCNT)"
results = cslulgtp('friday_status.', cartid,"1:30")
If friday_status.report.0 > 0 Then
Do
say friday_status.report.0
Do x = 1 to friday_status.report.0
say friday_status.report.x
End
End
```

The program would have results like this, where each line of the stem has a line of a formatted report.

```
6
Response for: QRY TRAN SHOW(PSB,QCNT)
Trancode MbrName CC PSBname QCnt LQCnt
ADDINV IMS2 0 0
ADDINV IMS2 0 DFSSAM04 2
ADDINV SYS3 0 DFSSAM04 1
ADDPART IMS2 0 0
```

Handling errors when using the CSLULGTP function

The CSLULGTP function will not set the "ctl.rc" and "ctl.rsn" stem variables if an error is encountered in the function itself. It is therefore highly recommended that any REXX program that uses the CSLULGTP function first check the IMSRC and IMSREASON REXX variables before any other processing continues to determine whether the function completed successfully.

If the IMSRC variable is nonzero, and the error was encountered in CSLULGTP itself, the value in IMSRC will begin with "08". In this case, the "ctl.rc" and "ctl.rsn" stem variables are not set and no data is returned in the stem variables.

The IMSRC and IMSREASON errors that can be returned are documented in the macro CSLUXRR.

For other errors (where the IMSRC variable is nonzero and does not begin with "08"), the "ctl.rc" and "ctl.rsn" stem variables will contain the command return code and reason code, and some of the other stem variables are set based on the command response. For example, if an invalid verb was entered, no command response data will be returned, but the "ctl.rc" and "ctl.rsn" stem variables as well as the REXX IMSRC and IMSREASON variables will be set.

REXX SPOC API within a transaction

Transactions can be written in the REXX language. A REXX EXEC runs as an IMS application and has characteristics similar to other IMS-supported programming languages. The REXX SPOC API can be used to issue commands from a transaction and to examine the command responses.

In the IMS Adapter for REXX environment, the OM API command authorization is performed with the MPP user ID. To use the transaction origin user ID rather than MPP user ID, a user exit must be utilized. The Build Security Environment user exit (BSEX) is called for non-OTMA/non-APPC input messages. The exit can request that IMS build a security environment in the MPP region when a message is scheduled (an accessor environment element (ACEE) will be anchored on TCBSENV). After use of the exit, the transaction origin user ID is used for OM API security checking. OTMA/APPC messages are not supported unless SECURITY=FULL or SECURITY=PROFILE, and the selection for the input message is FULL.

Related concepts

IMS Adapter for REXX reference (Application Programming APIs)

Ending the IMS SPOC environment

By issuing the END subcommand, you can end the IMS SPOC environment when you no longer want to execute IMS commands and signify that the SPOC environment is no longer needed.

Use the END subcommand to signify that the SPOC environment is no longer needed. After the END subcommand is issued, the control blocks associated with the SPOC environment are freed.

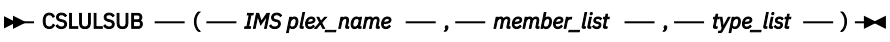
END is a valid IMS command. If you specify the END command with no operands, it is treated as an IMS SPOC subcommand. If you specify the END command with parameters, it is sent to the IMSplex for processing as an IMS command.

Retrieving unsolicited messages

After you set up the environment with CSLULXSB, you can use the CSLULSUB, CSLULUSB, and CSLULGUM functions to subscribe to OM for unsolicited output messages.

CSLULSUB request

By issuing the CSLULSUB request, you can start the subscription and monitor OM for unsolicited messages.



Parameters for the CSLULSUB request

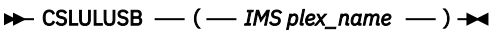
IMS plex_name
Name of the IMSplex. Do not use the 'CSL' prefix.

member_list
Specifies a list of member names of command processing clients in the IMSplex (such as IMS control regions) from which this OM client will receive unsolicited output messages. Do not specify this parameter if you specify the type list parameter. If you do not specify a member list or a type list, this OM client subscribes to all of the command processing clients in the IMSplex. Enclose the list in quotes and separate names with commas.

type_list
Specifies a type list of command processing client types (such as OM, RM, and IMS) from which this OM client will receive unsolicited output messages. Do not specify this parameter if you specify the member list parameter. If you do not specify either a member list or a type list, this OM client subscribes to all of the command processing clients in the IMSplex. Enclose the list in quotes and separate names with commas.

CSLULUSB request

By issuing the CSLULUSB request you can end the subscription. If a subscription does not exist, OM is no longer monitored for unsolicited messages.

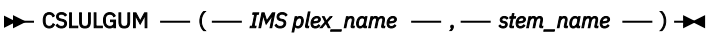


Parameters for the CSLULUSB request

IMS plex_name
Name of the IMSplex. Do not use the 'CSL' prefix.

CSLULGUM request

By issuing the CSLULGUM request you can copy unsolicited output messages to the REXX program variable named *stem_name*. The REXX program can then examine the messages and take appropriate action.



Parameters for the CSLULGUM request

IMS plex_name
Name of the IMSplex. Do not use the 'CSL' prefix.

stem_name
Name of a REXX variable that contains any unsolicited messages that are received. A stem variable is an array of values, with the '.0' element (for example 'mystem.0') indicating how many elements are in the array.

Sample program for subscribing to OM

This REXX SPOC API sample program uses CSLULSUB, CSLULBGUM, and CSLULUSB to subscribe to OM and to retrieve unsolicited messages. The program shows how you can invoke unsolicited output message functions.

```
/* rexx */
/*-------------------------------------------------------------
| REXX SPOC API example to invoke unsolicited output message  |
| functions.                                                  |
-------------------------------------------------------------*/

/*-------------------------------------------------------------
| tuning parameter: check every 10 seconds                    |
-------------------------------------------------------------*/
interval = 10

/*-------------------------------------------------------------
| We want to make syscalls, that is, sleep                    |
-------------------------------------------------------------*/
Call syscalls 'ON'

/*-------------------------------------------------------------
|  Establish IMS rexx environment                             |
-------------------------------------------------------------*/
Address LINK 'CSLULXSB'
If rc = 0 Then
 Do
   Address IMSSPOC
/*-------------------------------------------------------------
|  Subscribe to messages from IMSplex named  'PLEX1'          |
-------------------------------------------------------------*/
   continu = 1
   Do while(continu)
     subrc   = CSLULSUB('PLEX1')
     say 'subrc=('subrc')'
     If subrc = '01000010X' Then
       Do
          Say time()
        Address syscall "sleep" interval
       End
     Else
       continu = 0
   End


   Do a = 1 To 25
/*-------------------------------------------------------------
| wait a little before checking for new messages              |
-------------------------------------------------------------*/
     Address syscall "sleep" interval

/*-------------------------------------------------------------
|  Check if any unsolicited messages are present.             |
-------------------------------------------------------------*/
     results = CSLULGUM('PLEX1','xml.')
     say  'a='a  'results=('results')'

     If xml.0 /= '' Then
       Do
/*-------------------------------------------------------------
|  Display any messages in unsolicited message array.         |
-------------------------------------------------------------*/
         say 'xml.'0' = ('xml.0')'
         Do idx = 1 To  xml.0
           say 'xml.'idx'=('xml.idx')'
         End
```

```
      End
   End
/*-------------------------------------------------------------
|  Unsubscribe to unsolicited messages.                       |
-------------------------------------------------------------*/
   usbrc   = CSLULUSB('PLEX1' )

/*-------------------------------------------------------------
|  clean up REXX SPOC API                                     |
-------------------------------------------------------------*/
   "END"
 End
Exit
```

REXX samples and examples

These topics provide both sample programs and examples for REXX SPOC environments.

Sample REXX SPOC program

The following sample REXX program issues the IMS operator command **/DIS TRAN PART** and displays the command response.

Sample REXX program

```
Address LINK 'CSLULXSB'
    Address IMSSPOC
    /*-------------------------------------------------------------
    | 'ims' defines the IMSplex that receives the commands         |
    |                                                              |
    | 'route' defines which IMSplex members in the IMSplex         |
    | receives the commands.  If ROUTE is not specified or if      |
    | ROUTE * is specified, commands are routed to all IMSplex     |
    | members.                                                     |
    |                                                              |
    | 'wait' provides a timeout value to OM.  The time is in       |
    |  mmm:ss format (or ssss if no colon is specified).           |
    |                                                              |
    | 'cart' establishes the command response token for subsequent |
    | commands.                                                    |
    |                                                              |
    | 'end' frees control blocks                                   |
    |                                                              |
    -------------------------------------------------------------*/
    "IMS  IPLX4"
    "ROUTE IMS1,IMSB"

    "WAIT  5:00"

    "CART DISTRAN"
    "/DIS TRAN PART"

    /*-------------------------------------------------------------
     | The cslulgts function retrieves data associated with a       |
     | a specific token and fills in a REXX stem variable.  In      |
     | this example, it waits 59 seconds.                           |
     |                                                              |
     | The XML statements returned are put in the stem variable     |
     | specified by the user.                                       |
     |                                                              |
     -------------------------------------------------------------*/
     spoc_rc = cslulgts('DISINFO.','DISTRAN',"59")
     do z1 = 1 to  DISINFO.0
       /* display each line of XML information */
       Say disinfo.z1
     end
     "END"
```

REXX SPOC batch job example

These examples provide a sample batch job, a sample REXX SPOC program, and job output from the REXX SPOC example.

Sample REXX SPOC batch job

The batch job shown in the following figure calls the batch TSO command processor to get a response that contains all transactions that start with the letter V.

```
//REXXSPOC JOB ,
//   MSGCLASS=H,NOTIFY=USRT002,USER=USRT002,TIME=(,30)
//*
//SPOC     EXEC PGM=IKJEFT01,DYNAMNBR=45
//STEPLIB  DD DISP=SHR,DSN=IMS.SDFSRESL
//SYSPROC  DD DISP=SHR,DSN=LOCAL.IMS.CLIST
//SYSTSPRT DD SYSOUT=A
//SYSTSIN  DD *
  %REXXSPOC QRY TRAN NAME(V*)
/*EOF
```

The DD names in this batch job include:

STEPLIB
Contains the load modules.

SYSPROC
Contains the REXX programs.

SYSTSPRT
Contains the output produced by the REXX program.

SYSTSIN
Contains the command to execute, including its parameters.

The QRY TRAN command in the JCL is passed as an argument to the following sample REXX program. The command is issued, and the response is sent to the SYSTSPRT file.

REXX SPOC sample program

The following figure shows the sample REXX program, REXXSPOC.

```
/* rexx */
parse upper arg theIMScmd
Address LINK 'CSLULXSB'
if rc = 0 then
 do
   Address IMSSPOC
   "IMS plex1" ; if rc > 0 then say 'rc='imsrc 'reason='imsreason
   "route ims2"; if rc > 0 then say 'rc='imsrc 'reason='imsreason
   cartid = "TEST13"
   "cart" cartid ; if rc > 0 then say 'rc='imsrc 'reason='imsreason
   "WAIT 1:00" ; if rc > 0 then say 'rc='imsrc 'reason='imsreason
   theIMScmd
   if rc > 0 then say 'rc='rc 'imsrc='imsrc 'reason='imsreason
      do
        results = cslulgts('TEMP.', cartid,"1:30")
        say 'results='results ' imsrc='imsrc ' reason='imsreason
        if temp.0 /= '' then
          do
            say 'temp.'0'=('temp.0')'
            do idx = 1 to temp.0
              say 'temp.'idx'= 'temp.idx
            end
          end
      end
"END"
End
exit
```

Sample output

The output from the REXXSPOC sample program is shown in the following output example.

```
READY
%REXXSPOC QRY TRAN NAME(V*)
results=00000000X imsrc=00000000X reason=00000000X
temp.0=(30)
temp.1= <imsout>
temp.2= <ctl>
temp.3= <omname>OM1OM </omname>
temp.4= <omvsn>1.1.0</omvsn>
temp.5= <xmlvsn>1 </xmlvsn>
temp.6= <statime>2001.198 16:08:39.944953</statime>
temp.7= <stotime>2001.198 16:08:40.271944</stotime>
temp.8= <staseq>B625CACD49AF914A</staseq>
temp.9= <stoseq>B625CACD99848CC6</stoseq>
temp.10= <rqsttkn1>TEST13 </rqsttkn1>
temp.11= <rc>00000000</rc>
temp.12= <rsn>00000000</rsn>
temp.13= </ctl>
temp.14= <cmd>
temp.15= <master>IMS2 </master>
temp.16= <userid>USRT002 </userid>
temp.17= <verb>QRY </verb>
temp.18= <kwd>TRAN </kwd>
temp.19= <input>QRY TRAN NAME(V*)</input>
temp.20= </cmd>
temp.21= <cmdrsphdr>
temp.22= <hdr slbl="TRAN" llbl="Trancode" scope="LCL" sort="a"
key="1" scroll="no" len="8" dtype=" CHAR" align="left" />
temp.23= <hdr slbl="MBR" llbl="MbrName" scope="LCL" sort="a"
key="4" scroll="no" len="8" dtype="CHAR" align="left" />
temp.24= <hdr slbl="CC" llbl="CC" scope="LCL" sort="n"
key="0" scroll="yes" len="4" dtype="INT" align="right" />
temp.26= </cmdrsphdr>
temp.26= <cmdrspdata>
temp.27= <rsp>TRAN(VIDB ) MBR(IMS2 ) CC( 0) </rsp>
temp.28= <rsp>TRAN(VIDA ) MBR(IMS2 ) CC( 0) </rsp>
temp.29= </cmdrspdata>
temp.30= </imsout>
READY
END
```

/DISPLAY command examples and format identifiers

This example illustrates how format identifiers (FID) can be returned when the REXX program issues the operator commands, **/DISPLAY ACT** and **/DISPLAY STATUS**.

This example shows how format identifiers (FID) can be returned on operator commands. The REXX program issues two commands: **/DISPLAY ACT** and **/DISPLAY STATUS**.

```
Address LINK 'CSLULXSB'
if rc = 0 then
  do
    ADDRESS IMSSPOC
    "IMS  PLEX1"
    "CART  DISCART1"
    "DISPLAY ACT"
    RSP_RC1= CSLULGTS('DISCART1','ACT1.')
  End
```

In this example using **/DISPLAY ACT**, the command response includes the FIDs, because the default is to provide the FIDs.

In the example that follows using **/DISPLAY STATUS**, the CSLULOPT function is invoked before the command is issued. The CSLULOPT function specifies that FIDs are not to be included in the command response. In the command response, no FIDs are included.

```
OPT_RC = CSLULOPT('NOFID')
"CART  DISCART2"
"DISPLAY STATUS"
RSP_RC2= CSLULGTS('DISCART1','STAT1.')
```

Autonomic computing examples

These examples illustrate autonomic computing capabilities associated with the REXX SPOC API. Autonomic indicates that the code is responsive and can take certain actions to correct what it determines to be incorrect.

Autonomic example 1

In the following example, a transaction is queried. If the transaction is stopped, the REXX SPOC API attempts to start it. The REXX SPOC API examines the information returned by CSLULGTS, looking specifically for the line that refers to the transaction of interest.

```
/* autonomic computing example 1 */
"CART  qrytran12"
"qry tran name(CDEBTRN3) show(status)"
results = cslulgts("resp.","qrytran12","3:15")

Do idx = 1 to resp.0
  /* find which IMS and the status of tran */
  parse var resp.idx . "TRAN(CDEBTRN3" . ,
                        "MBR(" imsname ")" . ,
                       "LSTT(" status ")" .

  /* if tran is stoppped, try to start it */
  If pos('STOSCHD', status)  > 0 Then
    Do
      /* send command to IMS that needs to restart tran */
      "ROUTE"  imsname
      "UPD TRAN NAME(CDEBTRN3) START(SCHD)"
    End
End
```

Autonomic example 2

In the following example, the QUERY command is used to determine the queue count (qcnt) of a transaction. A qcnt with a value greater than 5 is considered a problem. The REXX SPOC API attempts to resolve the problem by starting another region and changing the transaction to a different class.

```
/* autonomic computing example 2  */
"CART  qrytran13"
"qry tran name(sks1) show(qcnt)"
results = cslulgts("resp.", "qrytran13", "3:15")
Do idx = 1 to resp.0
   parse var resp.idx . "TRAN(SKS1" . "Q(" count ")" .
   If count ¬= ''  &,
      count  > 5 Then
    Do
       "CART  strtrgn05"
       "START REGION IMSRG5"
       start? = cslulgts("strt.", "strtrgn05", "10:00")
       if imsrc = '00000000X' then
         Do
           "CART  updtran14"
           "update tran name(SKS1) set(class(5))"
         End
     End
End
"END"
```

Part 3. Asynchronous data propagation

You can propagate captured data asynchronously as an alternative to using the data capture exit routine. You can use this option to capture database changes for selected segment types and environment information in IMS data capture log records.

This alternative is available with the addition of a logging option on the EXIT= parameter of the DBDGEN utility. You can use the logging option to capture database changes for selected segment types and environmental information in IMS data capture log records. Captured information is compressed by using z/OS compression services to minimize the space that is needed to store the captured data on the IMS online data sets (OLDSs). After it is stored, the captured data is available for use; for example, to propagate to a Db2 for z/OS environment.

The data capture log records use the X'99' log code and have the following data capture subcodes to indicate the type of record that is being logged:

X'04'
Changed data

X'28'
End of job (EOJ)

X'30'
SETS call

X'34'
ROLS call

All data capture log records contain a common prefix, the Data Capture Log prefix, that contains the subcode that defines the type of record being logged. End of job records and SETS and ROLS call records consist entirely of this log record prefix and contain no additional information. Changed data log records, however, contain information in addition to the prefix and can span multiple physical log records to contain all of the captured data.

Related concepts
Data Capture exit routines (Database Administration)
Related reference
DBD statements (System Utilities)

Chapter 11. Changed data log record

To retrieve all the data logged for a particular call, you must examine the physical log records to determine where the logical log record begins and ends. You can examine the changed data log record to view the captured data captured for a DL/I call.

For each DL/I call, there is one logical log record that comprises all the data requested for that call. However, because all the data might not fit in one log buffer, one logical log record can actually be composed of several physical log records. To retrieve all the data logged for a particular call, you must examine the physical log records to determine where the logical log record begins and ends.

The data captured for a DL/I call is stored as elements within the changed data log record. For each DL/I call, there are multiple data elements recorded in the changed data log record. The details of the data elements are described in the following topics.

Elements of captured data

Each data element contains a 4-byte header with metadata the describes the type and length of the data that follows the header.

The captured data elements are logged in the DCAP_DATA field within the changed data log record. The format of data elements is broken down into the header information and the actual logged data. See “Format for data element header” on page 276 for the format of the log record header.

Each data element contains a 4-byte header. The header contains a 1-byte LOGID field describing the type of data being logged in this element, a 1-byte LOG_FLAG and a 2-byte LOG_LL field. The LOG_LL field contains the length of the data that follows the header. This length is used to locate the header for the next element. The header is followed by the actual data defined in the LOGID field. The data is compressed using the z/OS/ESA compression services, if such services are available. If so, the COMPR_ALGORITHM field in LOG_DCAP_DATA indicates the algorithm used to compress the data.

If FLAG_1 of the changed data log record contains X'60' or X'E0' (FIRST_RECORD and FIRST_CALL flags set), the next data element is LOG_INQY_DATA. This element is only present in the first physical record of the logical log record.

The type of data element is identified in the LOGID field of the header and can be one of the following values:

LOGID X'00'
There is a CAPD block for each EXIT= logging request.

LOGID X'04'
The DBD version for the data base, as specified on the VERSION= parameter of the DBD macro during DBDGEN. A character string of up to 63 characters.

LOGID X'08'
The concatenated key for the segment in the CAPD. The format of the physical concatenated key is a character string of up to 3,824 (255 x 15) characters.

LOGID X'0C'
The capture data segment data block (CAPD_DATA). This block is used for captured segment data, which can be path data, segment data, or before-image data. There is a CAPD_DATA block for each segment that was captured for the call. The blocks are logged in the following order: path data, segment data, before-image data (replace), twin data (insert).

LOGID X'10'
The segment data for the path or segment data deleted, inserted, or replaced (for a replace, this data would represent the after-image). The length of the segment data is a character string of up to 30,700 characters.

LOGID X'14'

The before-image of the segment data changed by the replace operation. Only the first changed byte through the last changed byte is logged. The unchanged beginning and ending portions of the segment are reconstructed from the after-image contained in segment data. The before-image data is a halfword offset value (the offset of the changed data within the segment) followed by a character string of up to 30,700 characters. Only the actual before-image is compressed. The halfword offset value is not compressed. Within the changed data log record, there can be one before-image for every changed segment.

If all the data elements do not fit in a single log record, the next log record (with the same PST_NUMBER) contains the remaining data elements, starting where the previous record left off. Before-image data, however, is unique in that the offset (2 bytes) of the data in the segment is logged preceding the changed data. Therefore, the length of the data in DCAP_DATA (LOG_LL) is 2 bytes greater than the actual length of the data logged. The offset field is not compressed.

Related reference

"CAPD_DATA format (LOGID=X'0C')" on page 281
The CAPD_DATA block contains fields that describe the name of the physical segment that is captured, the type of data being captured, the length and offset of the segment's key, and other information that pertains to the log record.

"CAPD block format (LOGID=X'00')" on page 278
The CAPD block contains fields that describe the name of the exit routine that is to be given control, the name of the physical database, the segment code, and other information that relates to the log record.

Reducing the amount of captured data

If the internal area of the changed data log record is not large enough to hold all of the data, the ERROR_LOG flag is set in the log record to indicate that the data is not complete. You can specify CASCADE,KEY,NODATA,NOPATH in the EXIT= parameter in the DBDGEN if cascade is required.

The changed data log record can contain a large amount of data for cascade deletes, especially when path data is requested. The data is staged in an internal area prior to logging. If this internal area is not large enough to hold all of the data, the ERROR_LOG flag is set in the log record to indicate that the data is not complete, although, the IMS data is committed and the call completes normally.

To avoid large amounts of data being written to the log and possible incomplete log records, specify CASCADE,KEY,NODATA,NOPATH in the EXIT= parameter in the DBDGEN if cascade is required. This specification results in just the concatenated key being written to the log during DLET operations, which significantly reduces the amount of data that is captured during a cascade delete. For data propagation to Db2 for z/OS where the primary key is derived from the concatenated key, the segment data is not required for the delete in Db2 for z/OS.

Example of logged data elements

Changed data log records contain many data elements as a result of a call. This topic provides an example where a third-level segment is replaced and path data is requested to be logged.

Changed data log records contain many data elements as a result of a call. Consider a situation where a third-level segment is replaced and path data is requested to be logged (Root segment is A, second-level segment is B, and the replaced segment is C). In this situation, the data elements are logged in the following order:

1. CAPD
2. DBD Version ID
3. Physical concatenated key
4. CAPD_DATA block for A
5. Segment data for A
6. CAPD_DATA block for B

7. Segment data for B
8. CAPD_DATA block for C
9. Segment data for C
10. CAPD_DATA block for C (before data)
11. Before-image data for C

If path data had not been requested, the CAPD_DATA blocks and segment data elements for A and B would not be logged.

Chapter 12. End of Job (EOJ) call log record

The EOJ call log record (X'28' subcode) is written when a batch DL/I program that has written changed data log records terminates normally. The record is written to indicate that the updates have been committed, because a commit record is not written to the log when a batch job terminates.

Related reference

“End of Job call log record format” on page 282

The end of job call log record contains various fields that describe the length of the record, the recovery token, and the CPU store clock time stamp.

Chapter 13. SETS and ROLS call log records

The SETS (X'30' subcode) and the ROLS (X'34' subcode) call log records are written whenever an application that might cause data to be captured issues a SETS or ROLS call using a token.

The log records are written to indicate that any changed data log records written after the SETS call for the token used in the ROLS call will have been aborted (backed out). The records are written even if exits are defined without a logging request.

Related reference

“SETS and ROLS call log record format” on page 282
The SETS and ROLS call log record contains various fields that describe the length of the record, the recovery token, and the CPU store clock time stamp.

Chapter 14. Format of the data capture log records

The topics in this section describe the format of the data capture log records.

Data capture log record prefix

The data capture log record prefix contains fields that describe the length, data capture subcode, recovery token, and CPU store clock time stamp of the log record.

The following table lists the prefix for data capture log records.

Table 69. Prefix for data capture log records

Field name	Data type	Field description
LL	H	The length of the log record, including an 8-byte log sequence number added by IMS to the end of the record
ZZ	H	Always zero
LOGCODE	XL1	X'99' log record code
SUBCODE	XL1	Data capture subcode
PST_NUMBER	H	PST number
RECOVTKN	XL16	The recovery token for the unit-of-recovery, which is used to associate the commit log records or the abort log records for this unit-of-recovery
STORE_CLOCK	XL8	The CPU store clock time stamp of the time the call completed and the log record was written

Changed data log record format

The changed data log record contains fields that describe the first and last log records written for the call, the user ID for the call, the data element that is being logged, and other information regarding the log record.

The following table lists the record formats for changed data log records.

Table 70. Format for changed data log records

Field name	Data type	Field description
FLAG_1	XL1	Flag 1: Bit definitions follow:
LAST_RECORD	X'80'	The last log record written for this call. If this bit is not on, the remaining data for this DL/I call is in the log records that follow.
FIRST_RECORD	X'40'	The first log record written for this call. If this bit is not on, the data logged at LOG_DATA_OFFSET is the data that would not fit in the preceding record.
FIRST_CALL	X'20'	The first log record written for this unit-of-recovery, so LOG_INQY_DATA is present in this log record. (FIRST_RECORD is on.)
	X'10'	Reserved for future use.
	X'08'	Reserved for future use.

Table 70. Format for changed data log records (continued)

Field name	Data type	Field description
ERROR_LOG	X'04'	This log record is not complete because there was an error during the processing of the log records for this call. Data for the call might not have been logged.
DBLEWORD_SEQNUM	X'02'	The log sequence number is a doubleword.
STCK_AT_END	X'01'	Store clock at end of the log record.
FLAG_2	XL1	Flag 2: Bit definitions follow:
V11_9904_FORMAT	X'80'	The CAPD_DATA blocks in this log record contain SEGMENT_RBA_64 and SEGMENT_RBA_64H.
V11_9904_PARTNM	X'40'	Partition name in the X'9904' log record.
V13_9904_DBVER	X'20'	The database version number is logged in this X'9904' log record.
V13_9904_POSDATA	X'10'	Positioning data is captured in this X'9904' log record.
LOG_DATA_OFFSET	H	The offset in the log record where the DCAP_DATA elements start. When FIRST_RECORD is not on (indicating a split log record), this field is the offset to the continuation of the data from the previous log record.
COMPR_ALGORITHM	XL1	The z/OS compression algorithm used to compress DCAP_DATA.
	XL1	Reserved for future use.
LOCK_SEQ_NUM	CL6	The IRLM lock sequence number used when IRLM SCOPE=GLOBAL used for block-level data sharing.
LOG_INQY_DATA		The following LOG_INQY_DATA fields are logged when FIRST_CALL is on.
PSBNAME	CL8	The application PSB name.
TRANNAME	CL8	The application transaction name.
USERID	CL8	The user ID for the call. For a batch job, if the JOB statement has the USER= keyword, the USERID is the RACF ID. For an online application, the USERID is either PSBNAME or RACFID.
RACF ID	CL8	The value from the USER= keyword from the JOB statement.
RACFID or PSB name	CL8	The value for online applications.
DCAP_DATA	CL0	The data element being logged.

Related concepts

z/OS: Data compression and expansion services

Format for data element header

The data element header contains fields that describe the type and length of data being logged as well as options for z/OS compression services.

The following table lists the format for data element headers.

Table 71. Format for log record header

Field name	Data type	Field description
LOGID	XL1	The type of data being logged. X'00'- CAPD block X'04'- DBD Version ID X'08'- Physical concatenated key X'0C'- CAPD_DATA X'10'- Segment data X'14'- Before-image data
LOG_FLAGS	XL1	Flags. Bit definitions follow:
COMPR_DATA	X'20'	The data is compressed using z/OS compression services.
	X'10'	Reserved for future use.
	X'08'	Reserved for future use.
	X'04'	Reserved for future use.
	X'02'	Reserved for future use.
	X'01'	Reserved for future use.
LOG_LL	H	Length of LOG_DATA.
LOG_DATA	CL0	Log data that is compressed if the z/OS services are available. The 4-byte data element header is followed by the type of data that is being logged: **X'00'** CAPD block **X'04'** The log data is the DBD Version ID. **X'08'** The log data is the physical concatenated key. **X'0C'** CAPD_DATA **X'10'** The log data is the Segment data. **X'14'** The log data is the Before-image data. Before-image data is preceded by a halfword offset of the changed data within the segment. The halfword offset is not compressed.

Related reference

“CAPD block format (LOGID=X'00')” on page 278
The CAPD block contains fields that describe the name of the exit routine that is to be given control, the name of the physical database, the segment code, and other information that relates to the log record.

“CAPD_DATA format (LOGID=X'0C')” on page 281

The CAPD_DATA block contains fields that describe the name of the physical segment that is captured, the type of data being captured, the length and offset of the segment's key, and other information that pertains to the log record.

CAPD block format (LOGID=X'00')

The CAPD block contains fields that describe the name of the exit routine that is to be given control, the name of the physical database, the segment code, and other information that relates to the log record.

The following table lists the CAPD block format.

Table 72. Format for CAPD block (LOGID = X'00')

Field name	Data type	Field description
NEXT_PTR	AL4	Internal use only. Not valid.
PREVIOUS_PTR	AL4	Internal use only. Not valid.
USER_EXIT_NAME	CL8	The name of the exit routine that is to be given control when an exit routine is requested in addition to logging. Blanks if the exit routine is not defined or is logged by DBCTL.
DATABASE_NAME	CL8	The name of the physical database.
DBD_VERSION_PTR	AL4	Internal use only. Not valid.
SEGMENT_NAME	CL8	The name of the physical segment that was updated and for which the log data was requested.
SEGMENT_CODE	XL1	Segment code for compare.
FLAGS	XL1	Flag: Bit definition follows:
DEDB_DB	X'80'	DEDB database.
KEY_NEEDED	X'40'	Concatenated key needed.
DATA_NEEDED	X'20'	Segment data needed.
PATH_NEEDED	X'10'	Path data needed.
	X'08'	Reserved for IMS.
	X'04'	Reserved for IMS.
	X'02'	Reserved for future use.
	X'01'	Reserved for future use.
	XL1	Reserved for IMS.
CALL_SEGMENT_LEVEL	XL1	Reserved for future use.

Table 72. Format for CAPD block (LOGID = X'00') (continued)

Field name	Data type	Field description
CALL_FUNCTION	CL4	Call function of request: **REPL** Replace call. **ISRT** Insert call. **DLET** Delete call. **FLD** Field call resulted in this update. Physical function will be "REPL". **CASC** Cascade delete as result of application delete call. **DLLP** Delete of a logical parent segment through its logical path because: • It was marked as previously deleted from its physical path. • It is vulnerable to delete from both the physical and logical paths. • The last logical child segment is being deleted. **Gxxx** When subset pointer updates are being captured, this is the get call (for example, GHU, GHNP) that was done.
PHYS_FUNCTION	CL4	Physical function performed by DL/I: **REPL** Segment physically replaced. **ISRT** Segment physically inserted. **DLET** Segment physically deleted. **DLPP** Delete this segment on its physical path, but do not physically remove because logical paths to the segment from a logical child still exist. **SSPU** SSP is updated when segment is physically retrieved or segment is not updated in path ISRT or REPL.
CALL_TIMESTAMP	XL8	STCK time stamp of call completion.
AREA_NAME	CL8	The AREA name for a DEDB database.
LOWEST_LVL_KEY_OR_DATA	XL1	The lowest level number for which path data or a part of the concatenated key is added.
	XL39	Reserved for future use.
COMMAND_CODES	0XL12	Command codes.

Table 72. Format for CAPD block (LOGID = X'00') (continued)

Field name	Data type	Field description
CMD_CODE_SNGL	XL1	Single-character command codes.
CMD_CODE_F	X'80'	Command code F.
CMD_CODE_L	X'40'	Command code L.
	XL1	Reserved for future use.
CMD_CODE_DBL	0XL10	Double-character command codes.
CMD_CODE_M	XL1	Subset pointer command codes M1 through M8. (*)
CMD_CODE_R	XL1	Subset pointer command codes R1 through R8. (*)
CMD_CODE_S	XL1	Subset pointer command codes S1 through S8. (*)
CMD_CODE_W	XL1	Subset pointer command codes W1 through W8. (*)
CMD_CODE_Z	XL1	Subset pointer command codes Z1 through Z8. (*)
	XL5	Reserved for future use
DB_VERSION	AL4	Database version number.
PART_NAME	CL8	HALDB partition name.
SAVED_DLTWA	AL4	Saved DLTWA for compare.
PHYSICAL_ROOT_RBA	AL4	RBA of root segment.
STORAGE_SIZE	F	Internal use only. Not valid.
CONC_KEY_PTR	AL4	Internal use only. Not valid.
CONC_KEY_LENGTH	H	Length of physical concatenated key.
ROOT_KEY_LENGTH	XL1	Length of root key.
CAPD_DATA_DIMENSION	XL1	Dimension of data array. This value reflects the number of CAPD_DATA Block elements that will be logged for this call. If path data is requested, the first CAPD_DATA block element will be for the root segment, followed by CAPD_DATA block elements for each segment in the path. The next CAPD_DATA block element (first if no path data) is for the segment associated with this request, followed by the CAPD_DATA block element (if any) associated with the before-image data (physical replace operations).
CAPD_DATA_PTR(16)	16AL4	Internal use only. Not valid.

(*): Each bit represents whether the corresponding command code number was specified. For example, if CMD_CODE_S is X'A0', it means that S1S3 was specified on the SSA.

Related concepts

"Elements of captured data" on page 267

Each data element contains a 4-byte header with metadata the describes the type and length of the data that follows the header.

CAPD_DATA format (LOGID=X'0C')

The CAPD_DATA block contains fields that describe the name of the physical segment that is captured, the type of data being captured, the length and offset of the segment's key, and other information that pertains to the log record.

The following table lists the format for CAPD_DATA.

Table 73. Format for CAPD_DATA (LOGID = X'0C')

Field name	Data type	Field description
NEXT_PTR	AL4	Internal use only. Not valid.
	CL4	Reserved for future use.
	AL8	Reserved for IMS.
SEGMENT_NAME	CL8	Physical segment name.
SEGMENT_LEVEL	XL1	Level of physical segment.
CMD_CODE_R	XL1	Subset pointer command codes R1 through R8. (*)
DATA_TYPE	XL1	The type of data being captured: **X'00'** Segment **X'01'** Before data **X'02'** Cascade data **X'03'** Segment path data **X'04'** Segment data of the twin that follows the segment being inserted.
DATA_FLAGS	XL1	Flag byte: Bit definitions follow:
DATA_USER_IO	X'80'	Data in user I/O area.
DELETED_ON_PHYS_PATH	X'40'	Segment is deleted on physical path.
	X'20'	Reserved for future use.
	X'10'	Reserved for future use.
	X'08'	Reserved for future use.
	X'04'	Reserved for future use.
	X'02'	Reserved for future use.
	X'01'	Reserved for future use.
LP_KEY_LENGTH	H	The length of the segment's logical parent key concatenated in front of the segment data. Zero if segment is not a logical child.
KEY_LENGTH	H	The length of the segment's key. Zero if the segment does not have a key.

Table 73. Format for CAPD_DATA (LOGID = X'0C') (continued)

Field name	Data type	Field description
KEY_OFFSET	H	The offset of the segment's key. Zero if the segment does not have a key.
SEGMENT_LENGTH	H	The length of the segment data.
SEGMENT_PTR	AL4	Internal use only. Not valid.

(*): Each bit represents whether the corresponding command code number was specified. For example, if CMD_CODE_R is X'20', it means that R3 was specified on the SSA.

Related concepts

"Elements of captured data" on page 267
Each data element contains a 4-byte header with metadata the describes the type and length of the data that follows the header.

End of Job call log record format

The end of job call log record contains various fields that describe the length of the record, the recovery token, and the CPU store clock time stamp.

The following table lists the end of job call log record formats.

Table 74. Format for EOJ call

Field name	Data type	Field description
LL	H	The length of the log record, including a 4-byte log sequence number added by IMS to the end of the record.
ZZ	XL2	Always zero.
LOGCODE	XL1	X'99' log record code.
SUBCODE	XL1	X'28' log subcode.
PST_NUMBER	H	PST number.
RECOVTKN	XL16	The recovery token for the unit-of-recovery, which is used to associate the commit log records or the abort log records for this unit-of-recovery.
STORE_CLOCK	XL8	The CPU store clock time stamp of the time that the program terminated and wrote the log record.

Related concepts

"End of Job (EOJ) call log record" on page 271
The EOJ call log record (X'28' subcode) is written when a batch DL/I program that has written changed data log records terminates normally. The record is written to indicate that the updates have been committed, because a commit record is not written to the log when a batch job terminates.

SETS and ROLS call log record format

The SETS and ROLS call log record contains various fields that describe the length of the record, the recovery token, and the CPU store clock time stamp.

The following table lists the format for the SETS and ROLS call log records.

Table 75. Format for SETS and ROLS call

Field name	Data type	Field description
LL	H	The length of the log record, including a 4-byte log sequence number added by IMS to the end of the record.
ZZ	XL2	Always zero.
LOGCODE	XL1	X'99' log record code.
SUBCODE	XL1	X'30' log subcode for SETS call; X'34' log subcode for ROLS call.
PST_NUMBER	H	PST number.
RECOVTKN	XL16	The recovery token for the unit-of-recovery, which is used to associate the commit log records or the abort log records for this unit-of-recovery.
STORE_CLOCK	XL8	The CPU store clock time stamp of the time the call completed and the log record written.
TOKEN	CL4	The SETS/ROLS token used by the call.

Related concepts

"SETS and ROLS call log records" on page 273

The SETS (X'30' subcode) and the ROLS (X'34' subcode) call log records are written whenever an application that might cause data to be captured issues a SETS or ROLS call using a token.

Chapter 15. Managing logging for multiple Data Capture exit routines for a single EXIT= parameter

If a single EXIT= parameter contains multiple Data Capture exit routines, and more than one of them has the LOG attribute specified, IMS writes the data to separate change capture log records in the IMS system log in the order that each exit routine that includes the LOG attribute is specified in the EXIT= parameter.

For example, if an EXIT= parameter is coded as follows:

```
EXIT=((PRODUCTA, KEY, DATA, CASCADE, LOG), (PRODUCTB, KEY, DATA, NONCASCADE, LOG))
```

when segments are inserted, updated, or deleted, IMS first records the change data per the exit routine specifications for PRODUCTA in one log record and then the change data for PRODUCTB's specifications in another.

Separate change capture log records are required, because the products that might consume the change capture log records may not be able to tolerate some of the data logged for the exit options of another product. Use of multiple records with different exit names, as specified in the EXIT= parameter, allows products to only process log records written for their product.

But excessive logging can impact the performance of certain IMS processes. Depending on the products that consume the change capture log records, however, you may be able to take steps to reduce the total amount of log data being written by IMS. If your products can tolerate extra information in the log data, you can try one of the following options.

Option 1

If the products can tolerate logged data they do not need, and one product requests all the information the other requires, then the LOG attribute only needs to be included for the product that requests the superset of data needed. For example:

```
EXIT=((PRODUCTA,KEY, DATA, CASCADE, LOG), (PRODUCTB, KEY, DATA, NONCASCADE, NOLOG))
```

In the preceding example, both the PRODUCTA and PRODUCTB exit routines will be called with the information they request, but only one change capture log record will be written for an update. Note that if PRODUCTB reads that record, it will need to consume it, even though the exit name is PRODUCTA, and tolerate the log data for the CASCADE information.

Option 2

If the products can tolerate logged data they do not need, but neither requests a superset of information, consider specifying NOLOG on all the named exit routines and adding one with all required options and LOG but no exit name, as in this example:

```
EXIT=((PRODUCTA, CASCADE, NOLOG), (PRODUCTB, KEY, DATA, NONCASCADE, NOLOG),
(*,KEY,DATA,CASCADE,LOG))
```

Here, the PRODUCTA and PRODUCTB exit routines will be called with the information they request, but there will be no change capture log records written using their exit names. Instead, there will be an unnamed record written that includes the superset of all data needed by both products. Note that in this example, if PRODUCTA reads that log record, it will need to ignore the KEY data, whereas PRODUCTB will need to ignore the CASCADE data.

Part 4. Database resource adapter (DRA)

The DRA is an interface to IMS DB full-function databases and data entry databases (DEDBs). Your application designer can create a program so that the DRA can be used by a coordinator controller (CCTL) or a z/OS application program that uses the Open Database Access (ODBA) interface.

These topics are intended for the designer of a CCTL or an z/OS application program. If you want more information about a specific CCTL's interaction with IMS DB or DB/DC, see the documentation for that CCTL.

Related concepts

Writing ODBA application programs (Application Programming)

Related tasks

Accessing IMS databases through the ODBA interface (Communications and Connections)

Chapter 16. Thread concepts

A DRA thread is a DRA structure that connects a CCTL task or a z/OS application program task with an IMS DB task that can process those calls. A single DRA thread is associated with every CCTL or ODBA thread. CCTL threads cannot establish a connection with more than one DRA thread at a time.

A *DRA thread* is a DRA structure that connects:

- A CCTL task (which makes database calls to IMS DB) with an IMS DB task that can process those calls. A CCTL thread is a CCTL task that issues IMS DB requests using the DRA.
- A z/OS application program task (which makes database calls to IMS DB) with an IMS DB task that can process those calls. An ODBA thread is a z/OS task that issues IMS DB calls using the DRA.

A single DRA thread is associated with every CCTL or ODBA thread. CCTL threads cannot establish a connection with more than one DRA thread at a time.

When the DRA Open Thread option is used, threads sign on to IMS when they are scheduled for a request. This behavior causes the IMS commands **/DISPLAY CCTL**, **/DISPLAY ACTIVE REGION** and **/DISPLAY ACTIVE THREAD** to show different values for the MINTHRD output field compared to a non-Open Thread system configuration. Because thread TCBs are not attached, MINTHRD does not apply to the number of threads attached at DRA initialization. However, it does indicate how many thread structures IMS allocates during initialization. After IMS begins scheduling PSBs, those threads appear in the command output as normal.

Processing threads

The way that the DRA processes a CCTL thread is different from how it processes an ODBA thread. In each case, the CCTL or z/OS application program issues a request through the creation of a DRA thread or the allocation of a DRA thread block, respectively.

Processing a CCTL thread

When a CCTL application program needs data from an IMS DB database, a CCTL task must issue a SCHED request for a PSB. To process the SCHED request, the DRA must create a DRA thread. To do this, the DRA chooses an available DRA thread TCB and assigns to it the CCTL thread token (a unique token that CCTL puts in the SCHED PAPL PAPLTTOK) and its own IMS DB task, which schedules the PSB. If the scheduling is successful, the DRA thread connection is considered complete because it now connects a CCTL thread to an IMS DB task using a specific DRA thread TCB.

Subsequent DRA requests from this CCTL thread must use the same CCTL thread token in order to ensure that the request goes to the correct DRA thread. When the application program finishes and the CCTL thread no longer needs the services of the DRA thread, the CCTL issues a TERMTHRD (Terminate Thread) request to remove the CCTL thread token from the DRA thread TCB and terminates the DRA thread. The thread TCB can then become part of a new DRA thread.

Processing an ODBA thread

When a z/OS application program needs data from an IMS database, an ODBA task must issue an APSB call to initialize the ODBA environment. To process the APSB call, the DRA allocates a DRA thread block and assigns to it the ODBA thread and its own IMS DB task, which schedules the PSB. If the scheduling is successful, the DRA thread connection is considered complete because it now connects an ODBA thread to an IMS DB task using a specific DRA thread block.

When the application program finishes and the ODBA thread no longer needs the services of the DRA thread, the ODBA application issues a DPSB call to terminate the DRA thread. The thread block can then become part of a new DRA thread.

Processing multiple threads

The ability of the DRA to process more than one thread at the same time is known as *multithreading*. Multithreading means that multiple CCTL or ODBA threads can be using the DRA at the same time. Multithreading applies to all DRA requests and ODBA calls.

Processing multiple CCTL threads

Multiple CCTL TCBs in a single address space can be used to process multiple CCTL threads. CCTL can dispatch each CCTL thread for a different CCTL TCB, and each CCTL TCB can call the DRA Startup/Router routine (DFSPRRC0) to process DRA requests.

To use the multithreading capability:

- The DRA must be initialized with more than one thread TCB. To initialize the DRA with more that one thread TCB, specify the MAXTHRD parameters (in the DRA Startup Table) as greater than one.
- The CCTL must be capable of processing its CCTL threads concurrently.
- The CCTL must have Suspend and Resume exit routines. The DRA uses these routines to notify the CCTL of the status of thread processing.

  Important: : The Suspend exit routine can start executing before or after the Resume exit routine to starts executing, but the Suspend exit routine cannot finish executing before the Resume exit routine starts executing. When you design the Suspend and Resume exit routines, ensure that the Suspend exit routine can determine whether the Resume exit has started or completed execution. If the Suspend exit routine determines that the Resume exit routine has not started executing, the Suspend exit routine must not return to the caller. If the Suspend exit routine determines that the Resume exit routine has started or completed execution, the Suspend exit routine should return to the Suspend exit caller and consider the suspend request complete.

Processing multiple ODBA threads

To use the multithreading capability, the DRA must be initialized with more than one DRA thread. To do this, specify the MAXTHRD parameters (in the DRA Startup Table) as greater than one.

CCTL multithread example

This example illustrates the concept of concurrent processing in a multithreading system.

Events in a multithreading system are shown in chronological order from top to bottom in the following example. To illustrate the concept of concurrent processing, the figure is split into two columns.

There are two CCTL threads and two DRA threads in the example. CCTLRTNA is the module name (for this example) of the CCTL routine that builds PAPLs and calls DFSPRRC0 to process DRA requests.

Important: In the following example, only one CCTL TCB is used. However, multiple CCTL TCBs in a single address space can be used to process multiple CCTL threads. CCTL can dispatch each CCTL thread for a different CCTL TCB, and each CCTL TCB can call the DRA Startup/Router routine (DFSPRRC0) to process DRA requests.

Table 76. Example of events in a multithreading system

CCTL TCB events	DRA TCB events
Application program1 needs a PSB, so CCTL thread1 is created. CCTL thread1 events: • CCTLRTNA builds the SCHED PAPL and calls DFSPRRC0.	

Table 76. Example of events in a multithreading system (continued)

CCTL TCB events	DRA TCB events
• DFSPRRC0 creates a DRA thread, and the thread token (PAPLTTOK) is assigned to DRA thread TCB1. • DFSPRRC0 activates thread TCB1. • DFSPRRC0 calls the Suspend exit routine. • The Suspend exit routine suspends CCTL thread1.	
	DRA thread TCB1 events: • The DRA processes the SCHED request and asks IMS DB to perform a schedule process. • Scheduling is in progress.
CCTL can now dispatch other CCTL threads for the CCTL TCB. Application program2 needs a PSB, so CCTL thread2 is created. CCTL thread2 events: • CCTLRTNA builds the SCHED PAPL and calls DFSPRRC0. • DFSPRRC0 creates a DRA thread, and a new thread token (PAPLTTOK) is assigned to DRA thread TCB2. • DFSPRRC0 activates thread TCB2. • DFSPRRC0 calls the Suspend exit routine. The Suspend exit routine suspends CCTL thread2.	
	DRA thread TCB2 events: • The DRA processes the SCHED request and asks IMS DB to perform a schedule process. • Scheduling is in progress.
Both threads are now suspended. The CCTL TCB is inactive until one of the threads resumes execution.	
	TCB2 scheduling finishes first. DRA thread TCB2 events: • Scheduling completes in IMS DB, and the PAPL is filled in with the results.

Table 76. Example of events in a multithreading system (continued)

CCTL TCB events	DRA TCB events
	• The DRA calls the Resume exit routine and passes the PAPL back to the CCTL. • The Resume exit routine sees the thread token (PAPLTTOK) and flags CCTL thread2 as 'ready to resume'. • The Resume exit routine returns to the DRA, and TCB2 becomes inactive.
Thread2 can resume immediately because the CCTL TCB is idle. Execution resumes directly after the point at which the thread was suspended by the Suspend exit routine.	TCB1 scheduling completes. DRA thread TCB1 events: • Scheduling completes in IMS DB and the PAPL is filled in with the results. • The DRA calls the Resume exit routine and passes the PAPL back to the CCTL.
Thread1 must wait until the Resume exit routine is available because thread2 has just resumed.	• The Resume exit routine sees the thread token (PAPLTTOK) and flags CCTL thread1 as 'ready to resume'. • The Resume exit routine returns control to the DRA and TCB1 becomes inactive.
CCTL thread2 events: • The Suspend exit routine returns to its caller, DFSPRRC0. • DFSPRRC0 returns to CCTLRTNA. • CCTLRTNA gets the results from the SCHED PAPL and gives them to the application program2. • CCTLRTNA finishes the thread2 SCHED request.	
After thread2 completes in CCTL TCB, the CCTL can dispatch thread1, which is currently waiting. CCTL thread1 events: • The Suspend exit routine returns to its caller, DFSPRRC0. • DFSPRRC0 returns to CCTLRTNA. • CCTLRTNA gets the results from the SCHED PAPL and gives them to the application program1. • CCTLRTNA finishes the thread1 SCHED request.	
CCTL thread2 events:	

Table 76. Example of events in a multithreading system (continued)

CCTL TCB events	DRA TCB events
• CCTLRTNA builds the DL/I PAPL and calls DFSPRRC0. • DFSPRRC0 finds the correct DRA thread. • DFSPRRC0 activates thread TCB2. • DFSPRRC0 calls the Suspend exit routine. • The Suspend exit routine suspends CCTL thread2. CCTL thread1 events: • CCTLRTNA builds the DL/I PAPL and calls DFSPRRC0. • DFSPRRC0 finds the correct DRA thread. • DFSPRRC0 activates thread TCB1. • DFSPRRC0 calls the Suspend exit routine. • The Suspend exit routine suspends CCTL thread1. Application program2 completes. The CCTL makes sync-point requests to IMS DB to commit the processing of this UOR. The CCTL flags the UOR for application program2 as in-doubt prior to issuing a phase 1 request. The CCTL keeps a record of this in-doubt UOR until the CCTL can make a successful phase 2 call to IMS DB.	
	DRA thread TCB1 events: • The DL/I call is in progress. • The DRA processes the DL/I PAPL and asks IMS DB to perform a DL/I process.
CCTL thread2 events: • CCTLRTNA issues a PREP request and calls DFSPRRC0. • DFSPRRC0 activates thread TCB2. • DFSPRRC0 calls the Suspend exit routine. • The Suspend exit routine suspends CCTL thread2.	
	DRA thread TCB2 events:

Table 76. Example of events in a multithreading system (continued)

CCTL TCB events	DRA TCB events
	• The DRA sends the PREP request to IMS DB. • IMS DB logs Phase 1 complete. This IMS DB UOR is now in-doubt. • The PREP request completes successfully and calls the Resume exit routine.
The PREP request results are returned to CCTLRTNA (by the PREP PAPL), and thread2 becomes inactive.	
	DRA thread TCB1 events: • IMS DB detects a DL/I failure. This failure results in termination of this DRA thread and a termination of this thread TCB. • Since this thread was in a schedule state, the DRA calls the Status exit routine and passes the DL/I PAPL back to the CCTL after putting UPSTOR information in it. • The Status exit routine associates UPSTOR with a CCTL thread, and the routine passes the PAPL to the DRA. • The DRA calls the Resume exit routine. • The DRA takes an SDUMP and terminates thread TCB1.
CCTL thread1 events: • After thread1 has been resumed, control is passed back to CCTLRTNA. • The CCTL notices that the DL/I request failed (returning PAPLRETC=4) and takes a diagnostic dump that includes UPSTOR. • The CCTL terminates this CCTL thread and frees UPSTOR because thread1 failed.	
Before CCTL sends a commit request for thread2, IMS DB fails.	
	DRA TCB events: • The DRA calls the Control exit routine to notify the CCTL that IMS DB failed. • The Control exit routine returns a code (PAPLRETC=8) that tells the DRA to reconnect to IMS DB.

Table 76. Example of events in a multithreading system (continued)

CCTL TCB events	DRA TCB events
	• The DRA terminates any existing thread TCBs. If the CCTL makes any subsequent requests to these terminated DRA threads, the DRA will respond with a return code indicating that the request cannot be processed. • After IMS DB has been restarted, the DRA successfully connects to IMS DB. • The DRA calls the Control exit routine to notify the CCTL that it successfully connected to IMS DB.
The CCTL creates a new task to resolve this in-doubt status because there is an entry in the resynchronization list for the IMS DB in-doubt UOR.	• The DRA passes the address of the resynchronization list (PAPLRST) to the CCTL. The list contains one entry for the IMS DB indoubt UOR for CCTL thread2. • The Control exit routine returns a code (PAPLRETC=0) that tells the DRA to continue running. • The DRA completes the setup process by creating new DRA thread TCBs
CCTL thread3 events: • The CCTL matches the in-doubt UOR in the re synchronization list with an in-doubt UOR in its own list. The CCTL in-doubt UOR is flagged for commit processing as its Phase 2 action. • CCTLRTNA issues a RESYNC request to DFSPRRC0 asking for commit processing. RESYNC is a DRA request, not a thread request. • DFSPRRC0 activates the DRA TCB to process the request and calls the Suspend exit routine.	
	• DRA calls IMS DB to commit its UOR. • After successful processing, DRA calls the Resume exit routine.
• After thread3 has been resumed, CCTLRTNA receives a return code of PAPLRETC=0. • The CCTL discards its indoubt UOR because the RESYNC request was successful.	

Chapter 17. Sync points

Sync point processing finalizes changes to resources. Sync point requests specify actions to take place for the resource changed (for example, commit or abort). A sync point is when IMS DB actually processes the request.

Each sync point is based on a unit of recovery (UOR). A UOR covers the time during which database resources are allocated and can be updated until a request is received to commit or abort any changes. Normally, the UOR starts with a CCTL SCHED (schedule a PSB) request or an ODBA APSB call and ends with a sync point request. Other DRA thread requests can also define the start and end of a UOR.

A CCTL UOR is identified by a recovery token (PAPLRTOK) that is received as part of a thread request that creates a new UOR. It is 16 bytes in length. The first 8 bytes contain the CCTL identification. This identification is the same as the CCTL ID that was a final DRA startup parameter determined from USERID or PAPLUSID in INIT request. The second 8 bytes must be a unique identifier specified by the CCTL for each UOR.

Related Reading: See the request descriptions under Chapter 24, "CCTL-initiated DRA function requests," on page 315 for more information on the DRA thread requests.

IMS DB expects the CCTL or the ODBA application to make the sync point decision and the resulting request. In the case of a CCTL, the CCTL is the sync point manager and coordinates the sync point process with all of the database resource managers (including those other than IMS DB) that are associated with a UOR. In the case of an ODBA application, z/OS Resource Recovery Services is the sync point manager and coordinates all the resource managers (including those other than IMS) that are associated with the UOR.

A CCTL working with a single resource manager may request a sync point in a single request or can use the two-phase sync point protocol which is required for a CCTL working with multiple resource managers. The single-phase sync point request can be issued when the CCTL has decided to commit the UOR, and when IMS DB owns all of the resources modified by the UOR.

An ODBA application must use the two-phase sync point protocol for committing changes to the IMS database.

Related reference

"SCHED request" on page 319
You can use the SCHED request to schedule a PSB in IMS DB. The first SCHED request made by a CCTL thread requires a new DRA thread. Existing DRA thread TCBs are used if they are not currently processing a DRA thread.

"SYNTERM request" on page 323
You can use the SYNTERM request to make a single-phase sync point request to commit the UOR or to release the PSB.

"PREP request" on page 324
You can use the PREP request to make a phase 1 sync-point request to ask IMS DB if it is ready to commit this UOR.

"COMTERM request" on page 325
You can use the COMTERM request to make a phase 2 sync-point request to commit the UOR or to release the PSB. You must issue a PREP request prior to issuing a COMTERM request.

"ABTTERM request" on page 326

You can use the ABTTERM request to make a phase 2 sync-point request to abort processing and release the PSB. The ABTTERM request does not require a preceding PREP request.

The two-phase commit protocol

The two-phase sync point protocol consists of two requests issued by the sync point manager to each of the resource managers involved in the UOR. Each of the UOR states, in-flight and in-doubt, define what happens to the UOR in the event of a thread failure.

Phase 1
The sync point manager asks all participants if they are ready to commit a UOR.

Phase 2
The sync point manager tells each participant to commit or abort based on the response to the request issued in phase 1.

A UOR has two states: in-flight and in-doubt. The UOR is in an in-flight state from its creation time until the time IMS DB logs the phase 1 end (point C in the following two tables). The UOR is in an in-doubt state from (point C) until IMS DB logs phase 2 (point D in the following two tables).

The in-doubt state for a single-phase sync point request is a momentary state between points C and D in Table 1.

The in-flight and in-doubt states are important because they define what happens to the UOR in the event of a thread failure. If a thread fails while its IMS DB UOR is in-flight the UOR database changes are backed out. If a thread fails when its IMS DB UOR is in-doubt, during single-phase commit, the UOR database changes are kept for an individual thread abend, but are not kept for a system abend. If a thread fails when its IMS DB UOR is in-doubt during two-phase commit, the database changes are kept.

Thread failure refers to either of these cases:

- Individual thread abends.
- System abends: IMS DB failure, CCTL failure, ODBA application failure, or z/OS failure (which abends all threads).

The following code shows the system events that occur when CCTL is used for single-phase sync point processing.

```
Time →
---A---B------C---D---E----
```

Table 77. CCTL single-phase sync point processing

Points In Time	System Events
A	CCTL phase 1 send
B	IMS DB phase 1 receive
C	IMS DB log phase 1 end
D	IMS DB log phase 2
E	CCTL phase 2 receive

The following table shows the system events that occur when CCTL is used for two-phase sync point processing.

```
Time →
---A---B-----C---D---E-----------F---G----H------J---K--------
```

Table 78. CCTL two-phase sync point processing

Points in time	System events
A	CCTL phase 1 send
B	IMS DB phase 1 receive
C	IMS DB log phase 1 end
D	IMS DB phase 1 respond
E	CCTL phase 1 receive
F	CCTL phase 2 send
G	IMS DB phase 2 receive
H	IMS DB log phase 2
J	IMS DB phase 2 respond
K	CCTL phase 2 receive

The following figure shows the system events that occur when two-phase sync point processing is completed using ODBA.

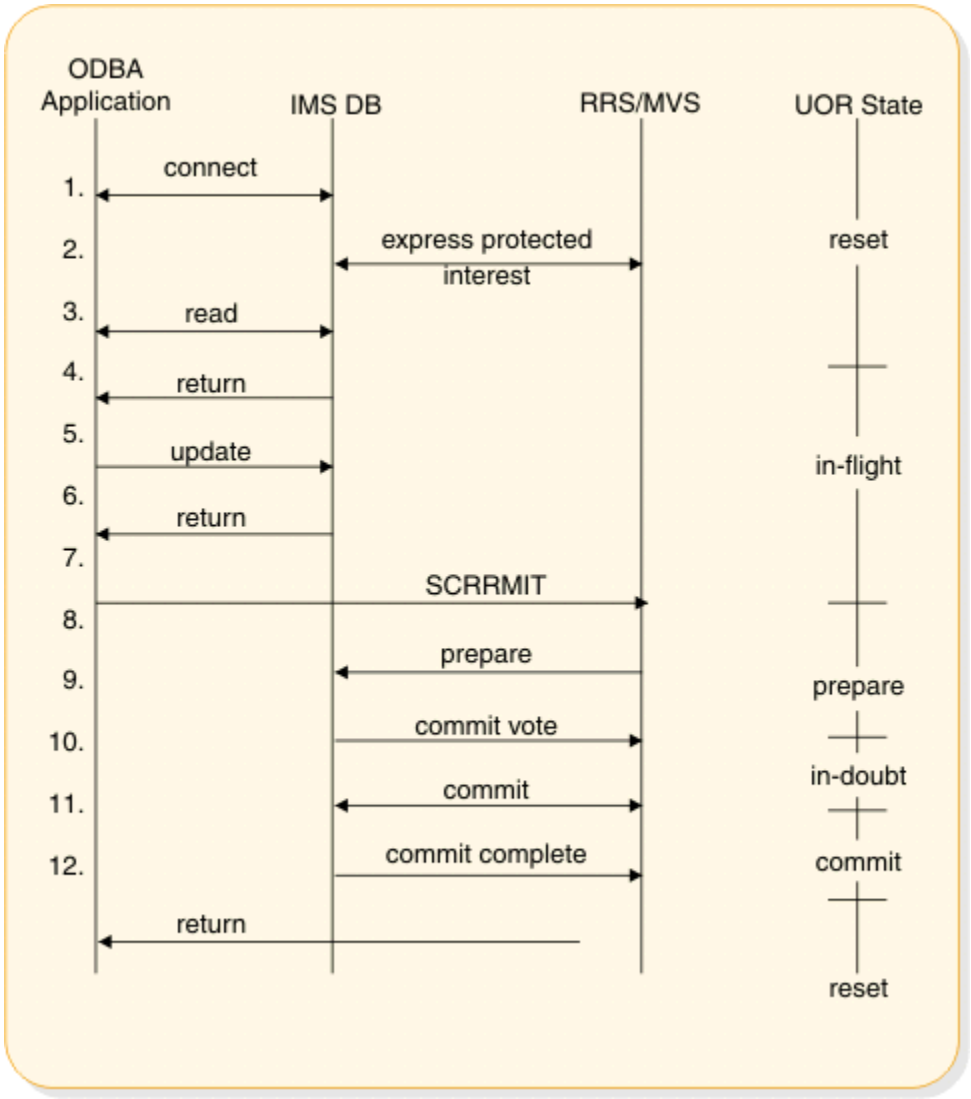


Figure 8. ODBA two-phase sync point processing

Note:

1. The ODBA application and IMS DB make a connection using the ODBA interface.
2. IMS expresses protected interest in the work started by the ODBA application. This informsz/OS Resource Recovery Services that IMS will participate in the two-phase commit process.
3. The ODBA application makes a read request to an IMS resource.
4. The ODBA application updates a protected resource.
5. Control is returned to the ODBA application following its update request.
6. The ODBA application requests that the update be made permanent by issuing the SRRCMIT call.
7. RRS calls IMS to do the prepare (phase 1) process.
8. IMS returns to RRS with its vote to commit.
9. RRS calls IMS to do the commit (phase 2) process.
10. IMS informs RRS that it has completed phase 2.
11. Control is returned to the ODBA application following its commit request.

In-doubt state during two-phase sync processing

A IMS DB UOR remains in the in-doubt state until a phase 2 request is received. This process is called "resolving the in-doubt". While a UOR is in-doubt, the database resources owned by that UOR are inaccessible to other requests. It is vital that in-doubts are resolved immediately.

CCTL example

If in-doubt UORs are created because IMS DB failed, the following sequence must occur to resolve the in-doubt UORs.

1. After restarting IMS DB, the CCTL should identify itself to IMS DB using an INIT request.
2. If identification is successful, the DRA notifies the CCTL control exit, passing to it a list of IMS DB UORs that are in-doubt.
3. The CCTL must resolve each in-doubt by making a RESYNC call, which causes a phase 2 action, commit or abort.
4. For CCTL to resolve a IMS DB in-doubt UOR, the CCTL must have a record of this UOR and the appropriate phase 2 action it must take. In this example, the CCTL record of a possible IMS DB in-doubt UOR is called a transition UOR.
5. The CCTL must define a transition UOR for the interval A-K (refer to Table 78 on page 299). Because this interval encompasses the IMS DB in-doubt period C-H, CCTL can resolve any in-doubts.

If a CCTL defines a transition UOR as interval E-K and if IMS DB fails while a thread is between C and D, IMS DB has an in-doubt UOR for which CCTL has no corresponding transition UOR, even though the phase 1 call failed. CCTL cannot resolve this UOR during the identify process. The only way to resolve this in-doubt is by using the IMS DB command, **/CHANGE CCTL**.

For ODBA, all in-doubts are resolved through z/OS Resource Recovery Services.

Chapter 18. DRA startup table

The database resource adapter (DRA) Startup Table contains values used to define the characteristics of the DRA. You must make the required changes to these modules to correctly specify the DRA parameters.

The DRA Startup Table is created by assembling:

- The DFSPZPxx module for a CCTL's use.
- The DFSxxxx0 module for ODBA's use.

The CCTL or ODBA system programmer must make the required changes to these modules to correctly specify the DRA parameters. The DRA parameters are specified as keywords on the DFSPRP macro invocation.

Sample DFSPZP00 source code

```
DFSPZP00 CSECT
         DFSPRP
DSECT=NO,FPBUF=10,FPBOF=5,CNBA=60,MINTHRD=3,MAXTHRD=6,DSNAME=IMS.SDFSRESL
         END
```

DFSPRP macro keywords

Keyword
Description

AGN=
This keyword is no longer used. If specified, it is ignored.

CNBA=

Specifies the total number of Fast Path NBA buffers for OBDM use. Valid values for CNBA are from 0 to 9999 or 1K to 32K. The default value for CNBA is 0.

You can determine a starting value for CNBA by using the following formula: (FPBUF+FPBOF) x MAXTHRDS = CNBA. The minimum value of CNBA must be equal to or greater than the total of the FPBUF and FPBOF values. If needed, adjust the value of CNBA to meet the performance and storage requirements of your installation.

If the FPBUF is greater than 0, FPBOF greater than 0, and CNBA equal to 0, the IMS system will calculate CNBA size during connection request processing based on the above formula.

DBCTLID=
The four-character name of the IMS DB or DB/DC region. This is the same as the IMSID parameter in the DBC procedure. The default name is SYS1.

DDNAME=
A 1-to-8 character ddname used with the dynamic allocation of the IMS DB execution library. The default ddname is CCTLDD.

DSNAME=
A 1-to-44 character data set name of the IMS DB execution library, which must contain the DRA modules and must be z/OS authorized. The default DSNAME is IMS.SDFSRESL. This library must contain the DRA modules.

FPBOF=
Specifies the number of Fast Path DEDB overflow buffers to be allocated per thread. Valid values for FPBOF are from 0 to 9999. The default value for FPBOF is 0. Values for FPBOF specified in the local section of CSLDCxxx override the value of FPBOF from the global section.

FPBUF=
Specifies the number of Fast Path DEDB buffers that are allocated and fixed per thread. Valid values for FPBUF are from 0 to 9999. The default value for FPBUF is 0. Values for FPBUF specified in the local section of CSLDCxxx override the value of FPBUF from the global section.

FUNCLV=
Specifies the DRA level that the CCTL or ODBA supports. The default is 3.

GENSNAP=
Specifies whether to produce or suppress SNAP output in DFSPAT20 during thread termination.

YES
Produces SNAP output in DFSPAT20 during thread termination. YES is the default.

NO
Suppresses the generation of SNAP output by DFSPAT20 during thread termination.

IDRETRY=
The number of times a z/OS application region is to attempt to IDENTIFY (or attach) to IMS after the first IDENTIFY attempt fails. The maximum number 255. The default is 0.

IMSPLEX=
A 1- to 5-character user-specified identifier that is concatenated to 'CSL' to create the z/OS cross-system coupling facility CSL IMSPLEX group name. The value specified here must match the IMSPLEX NAME= value specified in the SCI startup procedure. The IMS ODBA interface uses this XCF CSL IMSPLEX group to register with SCI using the CSLSCREG interface; the target ODBM address space must specify the same XCF CSL IMSPLEX group name when registering with SCI. If you specify this parameter, the IMS ODBA interface routes calls to the ODBM address space rather than directly to IMS. If IMSPLEX= is not specified, ODBA calls are routed directly to IMS.

This parameter is required for applications that route ODBA calls through the ODBM address space. There is no default value.

MAXTHRD=
The maximum number of DRA thread TCBs available at one time. The maximum number is 4095. The default is number 1.

MINTHRD=
The minimum number of DRA thread TCBs available at one time. The maximum number is 4095. The default is number 1.

When the DRA Open Thread TCB option is active, this value refers to the number of DRA threads that are signed on to IMS.

When the DRA Open Thread TCB option is inactive, this value refers to the number of DRA thread TCBs that remain attached and signed on to IMS.

ODBMNAME=
Specifies the name of the ODBM address space to which the ODBA interface registers using the CSLDMREG request, and to which the ODBA calls are routed using the CSLDMI interface. This is an optional parameter. There is no default. If ODBMNAME= is not specified, the ODBA interface selects which ODBM address space all ODBA calls are routed.

OPENTHRD=
Specifies whether or not to enable DRA Open Thread support processing. When enabled and using CICS® 4.2 or higher, this option directs the DRA not to attach dedicated IMS DRA thread task control blocks (TCBs). CICS TCBs are used instead, which are intended for increased parallelism within the CICS/DRA environment.

CCTL
Open Thread processing is enabled.

DISABLE
Open Thread processing is disabled. This is the default.

SOD=
The output class used for a SNAP DUMP of abnormal thread terminations. The default is A.

TIMEOUT=
(CCTL only). The amount of time (in seconds) a CCTL waits for the successful completion of a DRA TERM request. Specify this value only if the CCTL application is coded to use it. This value is returned to the CCTL upon completion of an INIT request.

TIMER=
The time (in seconds) between attempts of the DRA to identify itself to IMS DB or DB/DC during an INIT request. The default is 60 seconds.

TIMETHREADCPU=
Specifies whether the DRA monitors and reports the CPU usage statistics, that are related to DRA threads, when DRA Open Thread support is enabled. The time is reported in IMS 07 log record DLRTIME field, and to the CCTL in DRA thread statistics field PAPLCTM1.

YES
The DRA monitors and reports the CPU usage statistics. A value of YES overrides any value that is specified by the CCTL. YES is the default.

NO
The DRA does not monitor and report the CPU usage statistics. A value of NO overrides any value that is specified by the CCTL.

CLIENT
The DRA uses the setting that is specified by the DRA client through the INIT call (YES or NO) . If the client does not specify a setting, YES is the default.

USERID=
An 8-character name of the CCTL or ODBA region. The USERID name must not be the same as the IMSID name. This keyword is ignored for an ODBA Region.

Related concepts
Designing a DEDB or MSDB buffer pool (Database Administration)

Related reference
DBC procedure (System Definition)

Chapter 19. Enable the DRA for a CCTL

Two main steps are required to enable the DRA for a CCTL. After both steps are completed, the DRA is capable of handling other requests.

This section describes the two steps required to enable the DRA.

1. The coordinator controller (CCTL) system must load the DRA Startup/Router routine (DFSPRRC0) into a CCTL load library. Although DFSPRRC0 is shipped with the IMS product, it runs in the CCTL address space. Also, The version of the IMS DRA modules that are used by the CCTL must be the same version as the IMS with which the CCTL is communicating.

   Recommendations:

   - Concatenate the IMS.SDFSRESL library to the CCTL step library so the correct version of the DRA Startup/Router routine (DFSPRRC0) is loaded into the CCTL load library.
   - Ensure that the DRA Startup Table (DFSPZPxx) points to the correct version of IMS.SDFSRESL.
2. The system programmer must put the DFSPZPxx load module in a load library. The DRA is now ready to be initialized.

The CCTL starts the initialization process as a result of the CCTL application program issuing an initialization (INIT) request. At this point in time, the CCTL loads DFSPRRC0 and then calls the DRA to process the INIT request.

As part of the initialization request, the CCTL application program specifies the startup table name suffix (xx). The default load module, DFSPZP00, is in the IMS.SDFSRESL library.

After processing the INIT request, the DRA identifies itself to IMS DB. The DRA is then capable of handling other requests.

DFSPZP00 contains default values for the DRA initialization parameters. If you want to specify values other than the defaults, write your own module (naming it DFSPZPxx), assemble it, and load it in the CCTL load library. Use the supplied module, DFSPZP00, as an example.

The remainder of the DRA modules reside in a load library that is dynamically allocated by DFSPRRC0. The DDNAME and DSNAME of this load library are specified in the startup table. The default DSNAME (IMS.SDFSRESL) contains all the DRA code and is specified in the default startup table, DFSPZP00.

Related reference

Database resource adapter startup table for CCTL regions (System Definition)

“INIT request” on page 315

You can use the INIT request to initialize the DRA. The DRA startup parameter table contains all of the required parameters that you need to define the DRA. You can use the parameters given in the default module, DFSPZP00, or you can write your own module and bind it into the IMS.SDFSRESL data set.

Chapter 20. Enabling the DRA for the ODBA interface

Four steps are required to enable the DRA before an ODBA interface can use it. The first step is to create the ODBA DRA startup table.

There are four steps required to enable the DRA before an ODBA interface can use it:

1. Create the ODBA DRA Startup Table.
2. Verify that the ODBA and DRA modules reside in the STEPLIB or JOBLIB in the z/OS application region.
3. Link the ODBA application programs with DFSCDLI0.
4. Configure security.

The ODBA interface starts the initialization process after the ODBA application program issues either a CIMS INIT request or, if the ODBA application requires connections to multiple IMS systems, a CIMS CONNECT request. The CIMS INIT and CIMS CONNECT requests establish the ODBA environment in the address space. APSB requests then call the DRA to process the PSB schedule request with the IMS DB specified in DFS*xxxx*0, where *xxxx* is the DRA startup table name specified on the APSB call in the AIBRSNM2 field of the AIB.

After processing the CIMS INIT request or the CIMS CONNECT request, the DRA identifies itself to IMS DB if the optional xxxx value is passed on the CIMS INIT or CIMS CONNECT call. The DRA is then capable of handling other requests. The DRA's structure at this time is shown in the following figure.

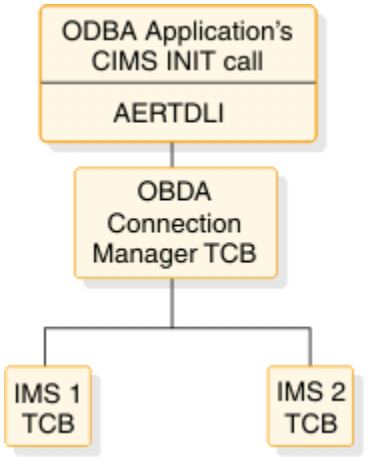


Figure 9. DRA component structure with the ODBA interface

The remainder of the DRA modules reside in a load library that is dynamically allocated by DFSAERA0. The DDNAME and DSNAME of this load library are specified in the startup table. The default DSNAME (IMS.SDFSRESL) contains all the DRA code.

Related reference

Database resource adapter startup table for CCTL regions (System Definition)

Chapter 21. Processing CCTL DRA requests

The CCTL communicates with IMS DB through DRA requests that are passed from the CCTL to the DRA using a participant adapter parameter list (PAPL). To make a DRA request the CCTL must pass control to the DRA Startup/Router Routine DFSPRRC0, and have register 1 point to a PAPL.

Multiple CCTL TCBs in a single address space can be used to process multiple CCTL threads. CCTL can dispatch each CCTL thread for a different CCTL TCB, and each CCTL TCB can call the DRA Startup/Router routine (DFSPRRC0) to process DRA requests.

Before passing control to DFSPRRC0, the CCTL must fill in the PAPL according to the request. These requests are specified by a function code in the PAPLFUNC field.

To specify a thread function request, put the PAPLTFUN value into the PAPLFUNC field.

The function requests are further broken down into many subfunctions. A thread function request is referred to by its subfunction name (for example, a thread request with a schedule subfunction is referred to as a SCHED request). Non-thread function requests are referred to by function name (for example, an initialization request is called an INIT request).

The term "DRA request" applies to both thread and non-thread function requests.

Once the PAPL is built and the DRA Startup/Router routine is loaded, the CCTL passes control to DFSPRRC0. The contents of the registers upon entry to DFSPRRC0 are:

Register
Contents

1
Address of the PAPL

13
Address of a standard 18-word save area

14
Return address of the calling routine

The DRA Startup/Router routine puts itself into 31-bit addressing mode and will return to the calling routine in the caller's original addressing mode with all its registers restored. Register 15 is always returned with a zero in it.

The return code for the request is in the PAPLRETC field of the PAPL.

Chapter 22. Processing ODBA calls

An ODBA application program communicates with IMS DB using the AERTDLI interface. The AERTDLI call interface processes DL/I calls from the ODBA application and also returns the results of those calls back to the ODBA using an AIB.

Unlike a CCTL's use of the PAPL, an ODBA application program communicates with IMS DB using the AERTDLI interface. The AERTDLI call interface processes DL/I calls from the ODBA application and also returns the results of those calls back to the ODBA using an AIB.

Related reference

Specifying the AIB mask for ODBA applications (Application Programming)

Chapter 23. Considerations for COMMIT CONTINUE-SYNC CONTINUE-ABORT CONTINUE

A 16-byte recovery token is used to uniquely identify a unit of recovery to IMS. This token is supplied by the CCTL.

The commit, sync, and abort continue behavior is used by the CCTL when multiple Units Of Recovery (UOR) are being performed within a single Unit Of Work (UOW). For CCTLs that employ COMMIT CONTINUE, SYNC CONTINUE, and ABORT CONTINUE behavior, the following recovery token information is strongly recommended by IMS. For commit and resolve indoubt processing, IMS passes the recovery token to identify the unit of work for which the requested action is to be taken.

The recovery token is constructed as shown in the following table:

Table 79.

CCTL-id	binary value	commit_number
8	5	3

where:

CCTL-id
Is the CCTL ID (1 to 8 characters).

binary value
Is a 5-byte binary origin application value assigned by the CCTL to the application when it is scheduled. The concatenation of CCTL and binary value (CCTL-id || binary value) is unique across the entire Unit of Work.

commit_number
Is a 3-byte binary commit number. The commit number should be initialized to binary zeroes when the application is scheduled and then incremented after each commit is processed for the application. The concatenation of CCTL-id, binary value, and commit number (CCTL-id || binary value || commit number) is unique across the Unit of Recovery.

The CCTL must ensure that the passed recovery token is unique. IMS only validates that the recovery token is unique, it does not verify the structure of the token.

The installation uses the **/DISPLAY CCTL** command to determine what units of work are in INDOUBT status in IMS. The installation can use the **/CHANGE** command (when necessary) to manually delete indoubt units of work in IMS. The **/CHANGE** command (when necessary) to manually delete INDOUBT units of work IMS. The **/CHANGE** command only affects unit of work status in IMS. There is no communication with the CCTL.

Chapter 24. CCTL-initiated DRA function requests

Certain requests are available to the CCTL that allow it to communicate with DBCTL. For all DRA requests, there are PAPL fields that the CCTL must fill in. The PAPLUSER field is a field to be used at the CCTL's discretion. One possible use for it is to pass data to exit routines.

This topic contains General-use Programming Interface information.

For all DRA requests, there are PAPL fields that the CCTL must fill in. When the DRA completes the request, there are some output PAPL fields that the DRA fills in. Some fields in the returned PAPL might contain the original input value.

(The PAPLTTOK and PAPLUSER fields retain the original input values.)

The PAPLUSER field is a field to be used at the CCTL's discretion. One possible use for it is to pass data to exit routines.

The DRA returns a code (in the PAPLRETC field) to the CCTL after processing a DRA request. The code indicates the status of the request and can be either an IMS code, a DRA code, or a z/OS code. Failed DRA requests return a nonzero value in the PAPLRETC field.

To use the default Suspend exit routine and Resume exit routine, each DRA request must have the field PAPLTECB set with the address of a CCTL ECB to be used if the thread is waited or posted. If your CCTL does not provide Suspend and Resume exit routines, the DRA default exit routines will be used.

Related concepts

"Problem diagnosis" on page 336
Diagnostic information is provided by the DRA in the form of an SDUMP, or a SNAP data set output. For X'80', the SDUMP is attempted first. If it fails, SNAP is done.

Related reference

DBCTL return codes (Messages and Codes)

INIT request

You can use the INIT request to initialize the DRA. The DRA startup parameter table contains all of the required parameters that you need to define the DRA. You can use the parameters given in the default module, DFSPZP00, or you can write your own module and bind it into the IMS.SDFSRESL data set.

The INIT PAPL also contains some parameters needed to initialize the DRA. If the same parameter appears in both the INIT PAPL and in the DRA startup parameter table, the specification in the INIT PAPL will override that in the startup table.

In addition to the required parameters of INIT PAPL, the optional parameters include:

Field
Contents

PAPLFUNC
PAPLINIT

PAPLSUSP
The address of the Suspend exit routine

PAPLRESM
The address of the Resume exit routine

PAPLTECB
To use the default Suspend exit routine and Resume exit routine, each DRA request must have the field PAPLTECB set with the address of a CCTL ECB to be used if the thread is waited or posted. If your CCTL does not provide Suspend and Resume exit routines, the DRA default exit routines will be used.

PAPLCNTL
The address of the Control exit routine

PAPLTSTX
The address of the Status exit routine

After the INIT request and the startup table have been processed, the DRA returns the following data to the CCTL in the INIT PAPL:

Field
Contents

PAPLDBCT
The IMS DB identifier (this is the IMSID parameter from system definition)

PAPLCTOK
The request token that identifies the CCTL to the DRA

PAPLTIMO
DRA TERM request timeout value (in seconds)

PAPLRETC
A code returned to the CCTL specifying the status of the request

PAPLDLEV
A flag indicating to CCTL which functions the DRA supports. (For the latest version of PAPL mapping format see the IMS. library; member name is DFSPAPL.)

INIT request, identify to DBCTL

To make the DRA functional, the DRA must identify itself to IMS DB, thus establishing a link between IMS DB and the CCTL. The identify process occurs in two cases:

- As a direct result of an INIT request.
- As part of a terminate/reidentify request from a Control exit routine invocation.

The DRA identifies itself to the IMS DB subsystem specified in the final DRA startup parameters. The identify process executes asynchronously to the INIT process. Therefore, it is possible for the INIT request to complete successfully while the identify process fails. In this case, the Control exit routine notifies the CCTL that the connection to IMS DB failed.

If IMS DB is not active, the console operator will receive a DFS690 message (a code of 0 was returned in the PAPLRETC field). You must reply with either a CANCEL or WAIT response. If you reply with WAIT, the DRA waits for a specified time interval before attempting to identify again. The waiting period is necessary because the identify process will not succeed until the DBCTL restart process is complete. You specify the length of the waiting period on the TIMER DRA startup parameter. If subsequent attempts to identify fail, the console operator will receive message DFS691, `WAITING FOR IMS DB`.

If the DRA cannot identify to IMS DB because the subsystem does not reach a restart complete state, there are two ways to terminate the identify process:

- The Control exit routine is called with each identify failure. This sets a PAPL return code of 4 or 8, terminating the identify process.
- The CCTL can issue a TERM request.

If you reply with CANCEL to message DFS690, control is passed to the Control exit routine, and the DRA acts upon the routine's decision.

After the identify process successfully completes, the DRA makes the CCTL address space non-swappable and calls the Control exit routine with a list of in-doubt UORs. If no in-doubt UORs exist, a null list is passed. The CCTL can use the RESYNC request to resolve any in-doubt UORs that do exist.

The INIT request attempts to create the MINTHRD number of thread TCBs. The actual number of TCBs created might be less than this value due to storage constraints.

INIT request after a previous DRA session termination

If a prior DRA session ended with a TERM request that received a PAPL return code=0, this INIT request must specify PAPLCTOK=0. If PAPLCTOK other than 0 is sent, the INIT request fails.

The INIT request must pass the PAPLCTOK value of the prior session in the current PAPLCTOK field if a DRA session ended because of:

- A nonzero return code from a TERM request.
- An internal TERM request from a Control exit routine request.
- A DRA failure.

INIT request to use the DRA open thread TCB option

The DRA open thread TCB allows the CCTL to direct the DRA not to attach dedicated DRA thread TCBs. Instead, DRA thread requests are processed on the CCTL application TCB.

The DRA open thread TCB option is either in use or not in use for the duration of the DRA instance.

To request activation of the DRA open thread TCB, set the following fields for the INIT request:

PALPFNCL
3 (PAPLFNC3)

PAPLOOTT
Set this bit (X’08’) to activate the open thread TCB.

You can verify that the TCB open thread is in use for the DRA by examining the PAPLDLEV flag field in the CCTL in INIT PAPL that the DRA returns after the INIT call. If the PAPLOTCB (X’08’) flag is set to 1, the open thread TCB option is in use.

INIT request to use the DRA open thread TCB option

When DRA open thread support processing is enabled, the DRA will or will not monitor and report the CPU time usage statistics that are related to the DRA thread processing based on the value specified for TIMETHREADCPU= on the DFSPRP macro in DFSPZPxx member.

If anything other that TIMETHREADCPU=CLIENT is specified in the DFSPZPxx member, the DRA ignores what is specified on the INIT call.

To request the DRA not to monitor CPU time usage statistics, set the following fields for the INIT request:

PALPFNCL
3 (PAPLFNC3)

PAPLOTCN
Set this bit (X'20') in field PAPLDROP to disable DRA thread CPU time usage statistics monitoring.

To determine if CPU usage statistics are being monitored for DRA thread processing, check the PAPLDLEV flag field in the CCTL INIT PAPL that the DRA returns after the INIT call. CPU usage statistics is not being monitored and reported if the PAPLOTCF (x'40') flag is set to 1.

INIT request to notify the DRA that CCTL is using Sync Continue behavior

This option directs IMS restart how to compare recovery tokens when there are multiple sync points in a single PSB schedule. To request activation of the DRA Sync Continue behavior and adherence to the IMS defined format of the recovery token, set the following fields for the INIT request:

PAPLOSCT
Set this bit (X'40') to activate the Sync Continue behavior.

Related concepts
“Enable the DRA for a CCTL” on page 305

Two main steps are required to enable the DRA for a CCTL. After both steps are completed, the DRA is capable of handling other requests.

RESYNC request

You can use the RESYNC request to tell IMS DB what to do with in-doubt UORs. Four different subfunction values indicate possible actions for IMS DB to take for the UOR.

The following subfunction values indicate possible actions:

PAPLRCOM
Commit the in-doubt UOR.

PAPLRABT
Abort the in-doubt UOR. Changes made to any recoverable resource are backed out.

PAPLSCLD
The UOR was lost to the transaction manager due to a coldstart.

PAPLSUNK
The in-doubt UOR is unknown to the CCTL. This can occur when the CCTL's in-doubt period does not include the start of phase 1. (See Table 78 on page 299 for an illustration of in-doubt periods.)

You must fill in the following input fields of the PAPL:

Field
Contents

PAPLCTOK
Request token

This token identifies the CCTL to the DRA. The DRA establishes the token and returns it to the CCTL in the parameter list on the startup INIT request. The request token must be passed on to the DRA for all RESYNC requests.

PAPLRTOK
Recovery token

This 16-byte token is associated with a UOR. The first 8 bytes must be the transaction manager subsystem ID. The second 8 bytes must be unique for one CCTL thread. This is one of the in-doubt recovery tokens passed to the Control exit routine.

PAPLFUNC
PAPLRSYN

PAPLSFNC
This field must contain PAPLRCOM, PAPLRABT, PAPLSCLD, or PAPLSUNK.

In addition to the required input parameters, the optional input parameters include:

Field
Contents

PAPLTECB
To use the default Suspend exit routine and Resume exit routine, each DRA request must have the field PAPLTECB set with the address of a CCTL ECB to be used if the thread is waited or posted. If your CCTL does not provide Suspend and Resume exit routines, the DRA default exit routines will be used.

TERM request

You can use the TERM request to terminate the IMS DB/CCTL connection and a remove the DRA from the CCTL environment. The DRA terminates after all threads have been resolved. No new DRA or thread requests are allowed, and current requests in progress must complete.

You must fill in the following input fields in the PAPL:

Field
Contents

PAPLFUNC
PAPLTERM, DRA terminate function code

PAPLCTOK
The DRA request token (output from an INIT request)

In addition to the required input parameters, the optional input parameters include:

Field
Contents

PAPLTECB
To use the default Suspend exit routine and Resume exit routine, each DRA request must have the field PAPLTECB set with the address of a CCTL ECB to be used if the thread is waited or posted. If your CCTL does not provide Suspend and Resume exit routines, the DRA default exit routines will be used.

After receiving the TERM request results, the CCTL might remove DFSPPRC0.

The fields returned in the PAPL to the CCTL are:

Field
Contents

PAPLRETC
The return code

PAPLMXNB
The number of times the maximum thread count was encountered during this DRA session

PAPLMTNB
The number of times the minimum thread count was encountered during this DRA session

PAPLHITH
The largest number of thread TCBs that were scheduled during this DRA session

PAPLTIMX
The elapsed time at maximum thread for this DRA session

SCHED request

You can use the SCHED request to schedule a PSB in IMS DB. The first SCHED request made by a CCTL thread requires a new DRA thread. Existing DRA thread TCBs are used if they are not currently processing a DRA thread.

If no TCBs are available, the DRA either creates a new thread TCB (until the maximum number of threads as specified by the MAXTHRD parameter in the INIT request is reached), or makes the SCHED request wait until a thread becomes available.

The value in the PAPLWCMD field indicates whether the thread to which the SCHED request applies is a short or long thread. The type of thread determines the action that IMS takes when a database command is entered for a database scheduled to the thread. The **/STOP DATABASE**, **/DBDUMP DATABASE**, or **/DBRECOVERY DATABASE** command issued against a database scheduled on a short thread will wait for the database to be unscheduled. IMS rejects these commands if they are entered for a database scheduled on a long thread.

You must fill in the following input fields in the PAPL:

Field
Contents

PAPLFUNC
PAPLTFUN, thread function code

PAPLSFNC
PAPLSCHE, schedule request subfunction code

PAPLCTOK
The DRA request token (output from an INIT request)

PAPLTTOK
The thread token set up by the CCTL

PAPLRTOK
The 16-byte UOR token (RTOKEN).

The first 8 bytes contain the CCTL identification. This identification is the same as the CCTL ID that was a final DRA startup parameter determined from USERID or PAPLUSID in INIT request; the USERID parameter is found in the DFSPRP macro used to generate the DFSPZPxx module. The second 8 bytes contain the unique identifier specified by the CCTL for each UOR.

PAPLPSB
The PSB name

PAPLWRTH
Deadlock Worth Value

If this thread hits a deadlock condition with any other DRA thread or with any IMS region, DBCTL collapses the thread with the lower deadlock worth value.

PAPLWCMD
This bit defines the thread as either a short or long thread which determines what action IMS takes on a **/STOP DATABASE**, /DBDUMP DATABASE, or **/DBRECOVERY DATABASE** command for a database scheduled to the thread.

If the bit is set on (X'80'), the database is scheduled on a short thread; if the bit is set off, the database is scheduled for a long thread.

PAPLFTRD
Fast Path Trace Option

If this bit is on (X'40'), Fast Path tracing in IMS DB is activated.

PAPLKEYP
Public Key Option

If this bit is set (X'10'), DBCTL builds UPSTOR area in a special subpool so that applications running in public key can fetch the UPSTOR area.

PAPLLKGV
Lockmax Option

If this bit is set (X'08'), DBCTL uses the value in PAPLLKMX as the maximum number of locks that this UOR can hold. Exceeding the maximum results in a U3301 abend.

PAPLLKMX
Lockmax Value, 0 to 255

This value overrides any LOCKMAX parameter specified on the PSBGEN for the PSB referenced in the SCHED request.

PALPUFXT
DRA open thread indicator.

Set PAPLUFXT to 1 for the SCHED call to indicate to the DRA that the thread TCB might not be the same TCB for the duration of the thread. The TCB might change between the time that the PSB is scheduled and the time that the PSB is unscheduled and the thread is terminated.

PAPLALAN
Application language type

Specifying the following input field is optional:

Field
Contents

PAPLSTAT
Address of an area where scheduled statistical data is returned to the CCTL.

If you do not want to allocate an address, enter 0.

PAPLPBTK
Address of the token for the z/OS Workload Manager performance block obtained by the CCTL.

You must specify this field for z/OS Workload Manager support for DRA threads.

PAPLTECB
To use the default Suspend exit routine and Resume exit routine, each DRA request must have the field PAPLTECB set with the address of a CCTL ECB to be used if the thread is waited or posted. If your CCTL does not provide Suspend and Resume exit routines, the DRA default exit routines will be used.

The output fields returned in the PAPL to the CCTL are:

Field
Contents

PAPLRETC
The return code

PAPLCTK2
The thread request token number 2. This is another DRA token required on future DRA requests originating from this thread.

PAPLPCBL
The address of the PCB list. There is one entry in the list for each PCB in the PSB that was scheduled, even if the PCB cannot be used with IMS DB.

The address of the PCB list is above the 16 MB line if either the PCBLOC=31 is set on the DFSPRP macro, or PAPLLPSO=31 is specified on the INIT request.

PAPL1PCB
The address of the PCBLIST entry pointing to the first database PCB

PAPLIOSZ
The size of the maximum I/O area

PAPLPLAN
The language type of the PSB

PAPLMKEY
The maximum key length

PAPLSTAT
The address of the schedule statistical data area. This address must be specified on the input field.

CCTLs currently using the IMS Database Manager and migrating to DBCTL will experience a change in the PCBLIST and user PCB area on a schedule request. The first PCB pointer in the PCBLIST contains the address of an I/O PCB. The I/O PCB is internally allocated during the schedule process in a DBCTL environment. The I/O PCB is normally used for output messages or to request control type functions to be processed. The PCBLIST and the PCBs reside in a contiguous storage area known as UPSTOR. If the PSB was generated with LANG=PLI, the PCBLIST points to pointers for the PCBs. If LANG= was not PLI, the PCBLIST points to the PCBs directly.

Related concepts

“Sync points” on page 297
Sync point processing finalizes changes to resources. Sync point requests specify actions to take place for the resource changed (for example, commit or abort). A sync point is when IMS DB actually processes the request.

“DRA tracing” on page 336

Tracing (logging) of activity does not occur in the DRA, but there is tracing in IMS DB of DL/I and Fast Path activity. The setup and invocation of DL/I tracing for IMS DB is the same as for IMS. The output trace records for CCTL threads contain the recovery token.

"CCTL performance: monitoring DRA thread TCBs" on page 333
You can evaluate the status of DRA thread TCBs from the output of the **/DISPLAY CCTL ALL** command in most cases. If there are no thread failures, the output might show fewer thread TCBs than the MINTHRD value because of internal short lived conditions.

IMS request

You can use the IMS request to make an IMS or Fast Path database request against the currently scheduled PSB.

You must fill in the following input fields in the PAPL:

Field
Contents

PAPLFUNC
PAPLTFUN

PAPLSFNC
PAPLDLI, DL1 request subfunction code

PAPLCTOK
DRA request token (output from an INIT request)

PAPLCTK2
Thread Token number 2. This is the DRA request token that is part of the output from a SCHED request.

PAPLTTOK
Thread token set up by the CCTL

PAPLRTOK
RTOKEN

A 16-byte UOR token.

PAPLCLST
The address of an IMS call list.

PAPLALAN
Application language type. This must reflect how the call list is set up. If PAPLALAN='PLI', the DRA expects the call list to contain pointers to the PCB's pointers. For any other programming language, the DRA expects direct pointers.

PAPLALAN does not have to match PAPLPLAN which schedules request returns. For example, if PAPLPLAN=PLI, the PCBLIST in UPSTOR points to an indirect list. If specified, the CCTL can use this to create a PCBLIST that application programs use. If the application programs are written in COBOL, the CCTL may create a new PCBLIST without pointers as long as the new list actually points to PCBs in UPSTOR. The application program IMS call lists can specify PAPLALAN=COBOL, and the DRA will not expect pointers in the call list.

In addition to the required input parameters, the optional input parameters include:

Field
Contents

PAPLTECB
To use the default Suspend exit routine and Resume exit routine, each DRA request must have the field PAPLTECB set with the address of a CCTL ECB to be used if the thread is waited or posted. If your CCTL does not provide Suspend and Resume exit routines, the DRA default exit routines will be used.

The output fields returned in the PAPL to the CCTL are:

Field
Contents

PAPLRETC
Code returned

PAPLSEGL
Length of data returned

Related concepts

"Sync points" on page 297
Sync point processing finalizes changes to resources. Sync point requests specify actions to take place for the resource changed (for example, commit or abort). A sync point is when IMS DB actually processes the request.

Related reference

Program communication block (PCB) lists (Application Programming)

SYNTERM request

You can use the SYNTERM request to make a single-phase sync point request to commit the UOR or to release the PSB.

You must fill in the following input fields in the PAPL:

Field
Contents

PAPLFUNC
PAPLTFUN

PAPLSFNC
PAPLSTRM, sync point commit/terminate subfunction code

PAPLCTOK
DRA request token (output from INIT request)

PAPLCTK2
The thread request token number 2. This DRA token is the output from the SCHED request.

PAPLTTOK
The thread token set up by the CCTL

PAPLRTOK
A 16-byte UOR token (RTOKEN).

You can specify the following, optional input fields:

Field
Contents

PAPLSTAT
Address of an area where transaction statistical data is returned to the CCTL.

PAPLTECB
To use the default Suspend exit routine and Resume exit routine, each DRA request must have the field PAPLTECB set with the address of a CCTL ECB to be used if the thread is waited or posted. If your CCTL does not provide Suspend and Resume exit routines, the DRA default exit routines will be used.

The output fields returned in the PAPL to the CCTL are:

Field
Contents

PAPLRETC
Code returned

PAPLSSCC
State of the single-phase sync point request at the time of the thread failure. This field is set if PAPLRETC is not equal to zero.

PAPLSTAT
The address of the transaction statistical data area. The address must be specified on the input field.

Related concepts

"Sync points" on page 297
Sync point processing finalizes changes to resources. Sync point requests specify actions to take place for the resource changed (for example, commit or abort). A sync point is when IMS DB actually processes the request.

PREP request

You can use the PREP request to make a phase 1 sync-point request to ask IMS DB if it is ready to commit this UOR.

You must fill in the following input fields of the PAPL:

Field
Contents

PAPLFUNC
PAPLTFUN

PAPLSFNC
PAPLPREP, sync-point prepare subfunction code

PAPLCTOK
DRA request token (output from an INIT request)

PAPLCTK2
Thread Token number 2. This is the DRA request token which is output from a SCHED request.

PAPLTTOK
The thread token set up by the CCTL

PAPLRTOK
A 16-byte UOR token (RTOKEN).

PAPLSDPL
A one-bit flag. Set this bit to 1 to indicate to IMS that this thread is part of a distributed unit of work.

In addition to the required input parameters, the optional input parameters include:

Field
Contents

PAPLTECB
To use the default Suspend exit routine and Resume exit routine, each DRA request must have the field PAPLTECB set with the address of a CCTL ECB to be used if the thread is waited or posted. If your CCTL does not provide Suspend and Resume exit routines, the DRA default exit routines will be used.

The following are output fields that are returned in the PAPL to the:

Field
Contents

PAPLRETC
Code returned

PAPLSTCD
Fast Path status code

If the value in the PAPLRETC field is decimal 35, the PAPLSTCD field contains a status code that further describes the error.

Related concepts

"Sync points" on page 297

Sync point processing finalizes changes to resources. Sync point requests specify actions to take place for the resource changed (for example, commit or abort). A sync point is when IMS DB actually processes the request.

COMTERM request

You can use the COMTERM request to make a phase 2 sync-point request to commit the UOR or to release the PSB. You must issue a PREP request prior to issuing a COMTERM request.

You must fill in the following input fields in the PAPL:

Field
Contents

PAPLFUNC
PAPLTFUN

PAPLSFNC
PAPLCTRM, sync-point commit/terminate subfunction code

PAPLCTOK
DRA request token (output from an INIT request)

PAPLCTK2
Thread Token number 2. This is the DRA request token that is output from a SCHED request.

PAPLTTOK
The thread token set up by the CCTL

PAPLRTOK
A 16-byte UOR token (RTOKEN).

Specifying the following input field is optional:

Field
Contents

PAPLSTAT
Address of an area where transaction statistical data is returned to the CCTL

PAPLTECB
To use the default Suspend exit routine and Resume exit routine, each DRA request must have the field PAPLTECB set with the address of a CCTL ECB to be used if the thread is waited or posted. If your CCTL does not provide Suspend and Resume exit routines, the DRA default exit routines will be used.

The output fields returned in the PAPL to the CCTL are:

Field
Contents

PAPLRETC
Code returned

PAPLSTAT
The address of the transaction statistical data area. This address must be specified on the input field.

Related concepts

"Sync points" on page 297

Sync point processing finalizes changes to resources. Sync point requests specify actions to take place for the resource changed (for example, commit or abort). A sync point is when IMS DB actually processes the request.

ABTTERM request

You can use the ABTTERM request to make a phase 2 sync-point request to abort processing and release the PSB. The ABTTERM request does not require a preceding PREP request.

You must fill in the following input fields of the PAPL:

Field
Contents

PAPLFUNC
PAPLTFUN

PAPLSFNC
PAPLATRM, sync-point abort/terminate subfunction code

PAPLCTOK
DRA request token (output from an INIT request)

PAPLCTK2
Thread Token number 2. This is the DRA request token, which is output from a SCHED request.

PAPLTTOK
The thread token set up by the CCTL

PAPLRTOK
A 16-byte UOR token (RTOKEN).

Specifying the following input field is optional:

Field
Contents

PAPLSTAT
Address of an area where transaction statistical data is returned to the CCTL.

PAPLTECB
To use the default Suspend exit routine and Resume exit routine, each DRA request must have the field PAPLTECB set with the address of a CCTL ECB to be used if the thread is waited or posted. If your CCTL does not provide Suspend and Resume exit routines, the DRA default exit routines will be used.

The output fields returned in the PAPL to the CCTL are:

Field
Contents

PAPLRETC
Code returned

PAPLSTAT
The address of the transaction statistical data area. This address must be specified on the input field.

Related concepts
“Sync points” on page 297
Sync point processing finalizes changes to resources. Sync point requests specify actions to take place for the resource changed (for example, commit or abort). A sync point is when IMS DB actually processes the request.

TERMTHRD request

You can use the TERMTHRD request to terminate the DRA thread.

You must fill in the following input fields of the PAPL:

Field
Contents

PAPLFUNC
PAPLTFUN

PAPLSFNC
PAPLTTHD, thread terminate subfunction code

PAPLCTOK
DRA request token (output from an INIT request)

PAPLCTK2
Thread Token number 2. This is the DRA request token that is output from a SCHED request.

PAPLTTOK
The thread token set up by the CCTL

Specifying the following input field is optional:

Field
Contents

PAPLSTAT
Address of an area where transaction statistical data is returned to the CCTL

PAPLTECB
To use the default Suspend exit routine and Resume exit routine, each DRA request must have the field PAPLTECB set with the address of a CCTL ECB to be used if the thread is waited or posted. If your CCTL does not provide Suspend and Resume exit routines, the DRA default exit routines will be used.

The output fields returned in the PAPL to the CCTL are:

Field
Contents

PAPLRETC
Code returned

PAPLSTAT
The address of the transaction statistical data area. This address must be specified on the input field.

Chapter 25. Terminating the DRA

Termination isolation means that a failure of the IMS DB subsystem does not cause a direct failure of any attached CCTL subsystem or ODBA application and vice versa.

Termination isolation should be one of your primary considerations when you design a CCTL subsystem or an ODBA application.

Although IMS DB was designed to prevent failure between connecting subsystems, a termination of a CCTL subsystem can cause IMS DB failure. If a DRA thread TCB terminates while IMS DB is processing a thread DL/I call on the CCTL's behalf, IMS DB fails with a U0113 abend.

The conditions that cause a thread TCB to terminate while IMS DB processes a DL/I call are:

- A DRA thread abend due to code failure. This can be corrected by fixing the failing code.
- The CCTL TCB collapses while a thread TCB still exists. The thread TCB collapses with an S13E or S33E abend and can result from three situations: a CCTL abend, a cancel command, or a shutdown.
- A DRA thread abend due to a IMS DB `/STOP REGION CANCEL` command initiated by CCTL.

An IMS DB U0113 abend can be prevented by designing the CCTL recovery process so that it issues a TERM request and waits for the request to complete. This allows the DRA and thread TCBs to terminate before the CCTL TCB terminates.

Related concepts

“Designing the CCTL recovery process” on page 331
You should consider CCTL operations and installation requirements when designing your CCTL. For example, a CCTL might have a means of allowing its own shutdown, but CCTL threads or BMPs should not have long-running UORs.

Chapter 26. Designing the CCTL recovery process

You should consider CCTL operations and installation requirements when designing your CCTL. For example, a CCTL might have a means of allowing its own shutdown, but CCTL threads or BMPs should not have long-running UORs.

Under the conditions of a nonrecoverable z/OS abend, a DRA TERM request lets all threads collapse and U0113 is possible. To reduce the number of nonrecoverable abends of the CCTL, IMS DB intercepts any operator CANCEL of a CCTL that is connected to IMS DB, and converts it to a S08E recoverable abend of the CCTL. If you want a CANCEL to be converted to an S08E abend, you must specify CCTCVCAN=Y on IMS startup parameters.

You can also as a last resort, force a CCTL to shut down. If an operator enters a FORCE command after CANCEL has been entered (and converted to S08E), IMS DB converts FORCE into a z/OS cancel command. Subsequent FORCE attempts are not intercepted by IMS DB. In these cases of nonrecoverable abends, a U0113 is possible.

A CCTL might have a means of allowing its own shutdown. The CCTL shutdown logic should issue a DRA TERM request and wait for the request completion to prevent a U0113 abend in IMS DB. The DRA TERM request waits for current thread requests to complete. One thing that can prevent a current thread DL/I call from completing normally is if the call has to wait in IMS DB for a database segment to become available. The reason the segment might not be available is that it is held by another UOR, either in a thread belonging to another CCTL or in an IMS dependent region (for example, a BMP). The solution is to not have CCTL threads or BMPs that have long-running UORs.

Recommendation: BMPs should take frequent checkpoints.

No matter how you choose to prevent or discourage long-running CCTL threads, you must decide how long to wait for the DRA TERM request to complete (TIMEOUT). In most cases, it is undesirable to get a U113 abend in IMS DB during a CCTL termination, so the timeout value should be greater than the longest possible UOR. If the CCTL has a means of limiting the UOR time or allowing the installation to specify this time limit, the DRA TERM timeout value can be determined. This timeout value can be specified in the DRA startup table and is returned to the CCTL in the INIT PAPL.

Recommendation: CCTL should use this DRA TERM timeout value when waiting for the DRA TERM request to complete. At the very least, by using the DRA TERM timeout value, you can control whether CCTL terminations cause IMS DB failures with respect to the UOR time length of the applications that run in a given IMS DB/CCTL session.

CCTL Operations Recommendation:

- Avoid using CANCEL or FORCE commands against CCTL regions that are connected to IMS DB.

CCTL Design Recommendations:

- The CCTL should issue a DRA TERM request during recoverable abend processing.
- CCTL shutdown functions should issue a DRA TERM request.
- Whenever a DRA TERM request is issued, wait for it to complete. If this time must have an upper limit, use the TIMEOUT value specified in the DRA startup table.
- The CCTL should prevent long-running UORs in its threads using IMS DB.

User Installation Recommendations:

- Have BMPs take frequent checkpoints.
- Limit long-running UOR applications.
- Set the TIMEOUT startup parameter as high as possible, preferably longer than the longest running UOR.

Related concepts

"Terminating the DRA" on page 329

Termination isolation means that a failure of the IMS DB subsystem does not cause a direct failure of any attached CCTL subsystem or ODBA application and vice versa.

Chapter 27. CCTL performance: monitoring DRA thread TCBs

You can evaluate the status of DRA thread TCBs from the output of the **/DISPLAY CCTL ALL** command in most cases. If there are no thread failures, the output might show fewer thread TCBs than the MINTHRD value because of internal short lived conditions.

Requirement: The DRA initialization process requires a minimum and maximum value (MINTHRD and MAXTHRD) for DRA thread TCBs. The value of MINTHRD and MAXTHRD determine the number of multithreading executions that can occur concurrently. These values also define the range of thread TCBs that the DRA will maintain under normal conditions with no thread failures. The number of TCBs can go below the MINTHRD value when the following thread failures occur:

- An abend.
- A nonzero DRA thread request return code that causes the thread TCB to collapse.
- Termination using a **IMS DB /STOP REGION** command.

Failed thread TCBs are not automatically recreated. The thread TCB number increases again if a new thread is created to process a SCHED request. If the number of thread TCBs is greater than the MINTHRD value and all thread activity ceases normally, the number of thread TCBs left in the DRA will be the MINTHRDD value.

During CCTL processing, the number of active DRA threads occupying thread TCBs varies from 0 to the MAXTHRD number. Active DRA threads indicate that at least one SCHED request has been made but not any TERMTHRD requests. If the number of non-active thread TCBs becomes too large, the DRA automatically collapses some thread TCBs to release IMS DB resources.

The status of DRA thread TCBs can be evaluated from the output of the **/DISPLAY CCTL ALL** command, except for one case.

If there were no thread failures, the output might show fewer thread TCBs than the MINTHRD value because of internal short lived conditions. In fact, the actual number of thread TCBs equals the MINTHRD.

Related concepts

z/OS: STIMER macro description

Related reference

“SCHED request” on page 319
You can use the SCHED request to schedule a PSB in IMS DB. The first SCHED request made by a CCTL thread requires a new DRA thread. Existing DRA thread TCBs are used if they are not currently processing a DRA thread.

/DISPLAY CCTL command (Commands)

DRA thread statistics

DRA thread statistics are returned for a SCHED request and for any DRA requests that terminate a UOR. The statistics are in a CCTL area that is pointed to by the PAPLSTAT field.

The PAPL listing maps this area, as shown in the following table. The statistics also appear in the IMS DB log records X'08' (SCHED) and X'07' (UOR terminate).

Table 80. Information provided for the schedule process

PAPL field	Field length (Hexadecimal)	Contents
PAPLNPSB	8	PSB name

Table 80. Information provided for the schedule process (continued)

PAPL field	Field length (Hexadecimal)	Contents
PAPLPOOL	8	Elapsed wait time for pool space (packed: microseconds)
PAPLINTC	8	Elapsed wait time - intent conflict (packed: microseconds)
PAPLSCHT	8	Elapsed time for schedule process (packed: microseconds)
PAPLTIMO	8	Elapsed time for DB I/O (packed: microseconds)
PAPLTLOC	8	Elapsed time for DI locking (packed: microseconds)
PAPLDBIO	4	Number of DB I/Os

Table 81. Information provided at UOR termination

PAPL field	Field length (Hexadecimal)	Contents
PAPLGU1	4	Number of database GU calls issued
PAPLGN	4	Number of database GN calls issued
PAPLGNP	4	Number of database GNP calls issued
PAPLGHU	4	Number of database GHU calls issued
PAPLGHN	4	Number of database GHN calls issued
PAPLGHNP	4	Number of database GHNP calls issued
PAPLISRT	4	Number of database ISRT calls issued
PAPLDLET	4	Number of database DLET calls issued
PAPLREPL	4	Number of database REPL calls issued
PAPLTOTC	4	Total number of DL/I database calls
PAPLTENQ	4	Number of test enqueues
PAPLWTEQ	4	Number of WAITS on test enqueues
PAPLTSDQ	4	Number of test dequeues
PAPLUENQ	4	Number of update enqueues
PAPLWUEQ	4	Number of WAITs on updates and enqueues
PAPLUPDQ	4	Number of update dequeues
PAPLEXEQ	4	Number of exclusive enqueues
PAPLWEXQ	4	Number of WAITs on exclusive enqueues
PAPLEXDQ	4	Number of exclusive dequeues
PAPLDATS	8	STCK time schedule started
PAPLDATN	8	STCK time schedule completed
PAPLDECL	2	Number of DEDB calls

Table 81. Information provided at UOR termination (continued)

PAPL field	Field length (Hexadecimal)	Contents
PAPLDERD	2	Number of DEDB read operations
PAPLMSCL	2	Reserved for Fast Path
PAPLOVFN	2	Number of overflow buffers used
PAPLUOWC	2	Number of UOW contentions
PAPLBFWT	2	Number of WAITs for DEDB buffers
PAPLUSSN	4	Unique schedule sequence number
PAPLCTM1	4	Elapsed UOR CPU time (for thread TCB)

DRA statistics

DRA statistics are contained in the returned PAPL as a result of a DRA TERM request, or in the Control exit routine's PAPL when it is called for DRA termination. This routine is called when the DRA fails or when a previous Control exit routine invocation resulted in return code 4.

The DRA statistics in the returned PAPL are:

1. The number of times the MAXTHRD value was reached.
2. The number of times the MINTHRD value was reached (only includes the times the value is reached when the thread TCB number is decreasing.)
3. The largest number of thread TCBs ever reached during this DRA session. (This is the number of TCBs, not the number of DRA threads, so it is at least the minimum thread value.)
4. The time (in seconds) during which the DRA thread TCB count was at the MAXTHRD value.

You can find the field names for the previous statistics in the PAPL extensions for the TERM PAPL and control exit routine PAPL.

Before attempting to evaluate the statistics DRA performance, remember:

- If the DRA is using the maximum number of threads (MAXTHRD), when the DRA receives any new SCHED requests it will make these requests wait until a thread is available.
- As active threads become available (for example, as a result of TERMTHRD call), some of the available threads might be collapsed.

These factors can adversely affect performance, but both improve IMS DB resource availability because fewer DRA threads require fewer IMS DB resources. The IMS DB resources (PSTs) are then available for other BMPs or other CCTLs to use.

Statistics 1, 2, and 4 can serve as a measurement of the two factors, and will help you decide how to balance performance and resource usage. For the sake of this discussion, these statistics are presented solely from a performance point of view (for example, assume only 1 CCTL connected to a IMS DB).

Evaluating the DRA statistics

If statistics 1 and 4 are high, a SCHED request had to wait for an available thread many times. To improve performance, raise the MAXTHRD value.

The impact of statistic 2 on performance can only be estimated if thread activity history is known (the DRA does not provide this history but the CCTL can). If activity is steady, little thread collapsing occurs and statistic 2 is meaningless. If activity fluctuates a lot, statistic 2 can be useful.

- If statistic 2 is 0, thread collapsing might have occurred, but the MINTHRD value was never reached.

- If statistic 2 is not zero, the MINTHRD value was reached and thread collapsing was stopped at those points, thus enhancing performance. Therefore, if you have highly fluctuating thread activity, you can improve performance by raising MINTHRD until statistic 2 has a nonzero value.

Finally, statistic 3 can be useful for adjusting your MAXTHRD value.

Note: These statistics are useful in determining MINTHRD and MAXTHRD definitions. When MINTHRD=MAXTHRD, these statistics are of no value.

DRA tracing

Tracing (logging) of activity does not occur in the DRA, but there is tracing in IMS DB of DL/I and Fast Path activity. The setup and invocation of DL/I tracing for IMS DB is the same as for IMS. The output trace records for CCTL threads contain the recovery token.

Fast Path tracing in IMS DB is different from IMS. Fast Path tracing in IMS DB is activated when a SCHED request to the DRA has the PAPLFTRD equal to ON (Fast Path trace desired for this UOR).

When this UOR completes, a trace output file is closed and sent to SYSOUT Class A.

If a thread request fails during Fast Path processing, the DRA might return the PAPL with the PAPLFTRR field equal to ON. This recommends to the CCTL that it request the PAPLFTRD field be equal to ON (Fast Path trace desired) in the SCHED PAPL if this failing transaction is run again by the CCTL.

Related reference

"SCHED request" on page 319
You can use the SCHED request to schedule a PSB in IMS DB. The first SCHED request made by a CCTL thread requires a new DRA thread. Existing DRA thread TCBs are used if they are not currently processing a DRA thread.

Sending commands to IMS DB

In an IMS DB warm standby or IMS XRF environment, a CCTL can use a z/OS SVC 34 to broadcast an emergency restart command to an IMS DB alternate, or a **SWITCH** command to an IMS XRF alternate in order to have the IMS alternate system become the primary IMS system.

These are the only IMS commands that can be done using this interface. The command verb can be preceded by either the command recognition character or the 4-character IMS identification that is in the PAPLDBCT field of the INIT PAPL.

Problem diagnosis

Diagnostic information is provided by the DRA in the form of an SDUMP, or a SNAP data set output. For X'80', the SDUMP is attempted first. If it fails, SNAP is done.

Failed DRA requests have a nonzero value in the PAPLRETC field of the PAPL returned to the CCTL. The format of PAPLRETC is:

```
HHSSSUUU
```

Where: HH= X'00'- No output

UUU
IMS DB return codes

X'88'- No output

SSS
All z/OS non-retrievable abend codes (for example, 222, 13E) or,

UUU
IMS abend codes (775, 777, 844, 849, 2478, 2479, 3303)

X'84'- SNAP only

UUU
IMS abend codes (260, 261, 263)

X'80'- SDUMP/SNAP provided

SSS
All the z/OS abend codes that can be tried again

UUU
All IMS abend codes besides those that are listed for the format of PAPLRETC

Diagnostic information is provided by the DRA in the form of an SDUMP, or a SNAP data set output. For X'80', the SDUMP is attempted first. If it fails, SNAP is done. For X'84', no SDUMP is attempted, but a SNAP is attempted.

A z/OS or IMS abend code failure results in DRA thread termination and thread TCB collapse. An IMS DB return code has no affect on the DRA itself or the thread TCB.

DRA thread TCB failures that occur when not processing a thread request result in a SDUMP/SNAP process. DRA control TCB failures that occur when not processing a DRA request result in a SDUMP/SNAP process and the Control exit routine is called. For a thread request of type SCHED, a failure with X'80' or X'84' can result in either SNAP or SDUMP.

SDUMP

SDUMP output contains:

- The IMS control region.
- DLISAS address space.
- Key 0 and key 7 CSA.
- Selected parts of DRA private storage, including the address space control block (ASCB), task control block (TCB), and request blocks (RBs).

You can format the IMS control blocks by using the Offline Dump Formatter (ODF).

The ODF will not format DRA storage. You can use IPCS to format the z/OS blocks in CCTL private storage.

DRA SDUMPS have their own SDUMP options. As a result, any CHNGDUMP specifications cannot cause sections of DRA SDUMPs to be omitted. If these specifications are not in the DRA list of options, they can have an additive effect on DRA SDUMPS.

SNAP

The SNAP dump data sets are dynamically allocated whenever a SNAP dump is needed. A parameter in the DRA Startup Table defines the SYSOUT class.

The SNAP output contains:

- Selected parts of DRA private storage, including the ASCB, TCB, and RBs.
- IMS DB control blocks.

Related reference
Offline Dump Formatter utility (DFSOFMD0) (System Utilities)

Part 5. Database Recovery Control (DBRC)

You can use DBRC to record and manage information that is stored in a set of VSAM data sets that are collectively called the RECovery CONtrol (RECON) data set. Based on this information, you can use DBRC to advise IMS about how to proceed for certain IMS actions.

Chapter 28. DBRC API

Your applications can obtain services from Database Recovery Control (DBRC) through the DBRC application programming interface (API), a release-independent, assembler macro interface. The application obtains these services by issuing DBRC API requests to DBRC, and DBRC returns the results to an area in storage where the application can retrieve them.

The DBRC API is provided with IMS in the DSPAPI macro. A sample application program (DSPAPSMP) that uses the DBRC API is included in the IMS.ADFSSMPL (also known as IMS.SDFSSMPL) library.

Important: All DBRC API requests must be issued under the same TCB where the DBRC start request (STARTDBRC) was issued. Any request that is issued under a different TCB fails with reason code X'C900000A'.

To write a program that uses the DBRC API, you must have a working knowledge of:

- Assembler language programming
- z/OS and the services it supplies
- IMS
- DBRC

Related concepts

z/OS: HLASM Language Reference

z/OS: HLASM Programmer's Guide

"Output from query requests" on page 368
Requested information is returned to the calling application in a chain of one or more blocks in storage. The pointer to the beginning of this chain is returned in the area specified by the OUTPUT parameter of the Query request.

Related reference

"DBRC authorization request (AUTH)" on page 353
You can use the DBRC authorization request to ensure that an invalid data sharing environment is not created. Database authorization is the process of requesting permission to access and use a database. In this context, the database is either a DL/I database or a Fast Path DEDB area.

"DBRC command request (COMMAND)" on page 361
You can use the DSPAPI FUNC=COMMAND request to issue a DBRC utility command from your application program. All DBRC commands are accepted on this request, except for the LIST command.

"DBRC query request (QUERY)" on page 367
You can use the DBRC Query request (DSPAPI FUNC=QUERY) along with the TYPE parameter to retrieve various types of information from the RECON data set.

"DBRC release buffer request (RELBUF)" on page 437
You can use the DBRC release buffer request (DSPAPI FUNC=RELBUF) to release storage that was acquired as a result of DBRC query, command, authorization, and unauthorization requests.

"DBRC start request (STARTDBRC)" on page 439
You can use the DBRC start request (STARTDBRC) to initialize the DBRC API and to start DBRC.

"DBRC stop request (STOPDBRC)" on page 443
You can use the STOPDBRC request to terminate the DBRC application and to stop DBRC.

"DBRC unauthorization request (UNAUTH)" on page 445
You can use the UNAUTH request to explicitly remove authorization to a database or area. Authorization by an application is implicitly removed by the STOPDBRC request. UNAUTH is the opposite of FUNC=AUTH.

"Database query request (TYPE=DB)" on page 373

You can use the Database Query request (DSPAPI FUNC=QUERY TYPE=DB) to retrieve information from the RECON concerning one or more registered databases.

"DBDS query request (TYPE=DBDS)" on page 394
You can use the DSPAPI FUNC=QUERY TYPE=DBDS request to retrieve information from the RECON data set for one or more DBDSs in a non-HALDB database, a HALDB partition, a DBDS group, or a CA group. You can also request recovery related information for the data set, including allocation, image copy, recovery, and reorganization information.

Structure of applications that access the DBRC API

Your applications must follow a general structure in order to access the DBRC API.

The general structure of an application that uses the DBRC API is as follows:

1. Include the API DSECTS (DSPAPI FUNC=DSECT)
2. Initialize the API, start DBRC, establish a connection to the RECON data sets, and receive the API token (DSPAPI FUNC=STARTDBRC)
3. Issue one or more Query requests (DSPAPI FUNC=QUERY)
4. Process the information that is returned from the Query requests
5. Return buffer storage (DSPAPI FUNC=RELBUF)
6. Terminate DBRC and the DBRC API (DSPAPI FUNC=STOPDBRC)

How an application program establishes the DBRC API environment

Your application program establishes the application programming interface (API) environment by using the DSPAPI FUNC=STARTDBRC macro. This macro function initializes the API, creates a connection to the Database Recovery Control (DBRC) RECON data sets, and returns the API token.

Requirements:

- The API token must be specified for all subsequent API calls.
- All API requests must be issued under the same task control block (TCB) where the FUNC=STARTDBRC was issued.

Related concepts

"The DBRC API token" on page 344
The DBRC API token is a 4 byte field that is used to relate a series of API requests.

How an application program ends the DBRC API environment

After it has completed its work with DBRC, your application program must end the API environment using the DSPAPI FUNC=STOPDBRC macro. This macro function allows DBRC to terminate, deallocates any data sets that DBRC had allocated, and frees storage allocated to the API environment.

Addressing and residency mode

Your application must invoke the DBRC API in 31-bit addressing mode and can reside either above or below the 16 MB line.

Parameter addresses passed to the API can be above or below the 16 MB line unless otherwise stated in a macro description. Parameter addresses returned to the user by the API likewise can be above or below the 16 MB line.

Address space control (ASC) mode and state

Your application must be in primary ASC mode to invoke the API services.

Cross memory mode is not supported. The application can run in either problem program state or supervisor state, and can also run as an APF authorized program.

How the DBRC API uses registers

General purpose registers 0, 1, 14, and 15 can be changed by the DBRC API. Register 13 must contain the address of a standard (18 word) save area. The contents of registers 2-13 are unchanged by the DBRC API.

How to include equate (EQU) statements in your DBRC API application

You can use the DSECT function of a DSPAPI request to include equate (EQU) statements and constants used by the DSPAPI macro in your program.

The following syntax is for the DSECT function of a DSPAPI request.

API application

Parameters that are specified on the DSPAPI macro are either literals, addresses, or values. You can pass an address or a parameter value by using a register, a symbol, or a literal. You can specify literal values in mixed case.

Use a register

To use a register, you must load the address of a symbol or value into one of the general purpose registers, and then use that register (enclosed in parentheses) for the parameter in the DSPAPI request. Use only registers in the range 2-12. Register notation does not work with MF=L because this form does not produce executable code.

The following example shows how to pass an address using a register:

```
         LA 5,OUTPUTAD
         DSPAPI FUNC=QUERY,OUTPUT=(5),...
     .
     .
     .
OUTPUTAD     DS     A
```

The previous example generates the following instruction:

```
ST    5,DSPAPI_Plist_Output
```

Use a symbol name

To use a symbol name, you must define a named area of storage that either contains the desired value, or will receive a returned address or value, and then use that symbol name for the parameter in the DSPAPI request.

The following example shows how to pass an address using a symbol name:

```
DSPAPI FUNC=QUERY,OUTPUT=OUTPUTAD,..
     .
     .
     .
OUTPUTAD     DS     A
```

The previous example generates the following instructions:

```
LA    0,OUTPUTAD
ST    0,DSPAPI_Plist_Output
```

Use a literal

You can use literals for certain parameter values, such as time stamps. To use a literal, pass the literal as a hexadecimal string for the parameter in the DSPAPI request. Literals can also be mixed-case.

The following example shows how to pass a value using a literal:

```
                                                72
DSPAPI FUNC=QUERY,TYPE=LOG,                     C
       STARTIME==XL12'1980030F191212009999028D'
```

The previous example generates the following instructions:

```
LA     0,=XL12'198030F191212009999028D'
ST     0,DSPAPI_Plist_Startime
```

Unless specifically noted, name fields are 8 characters long, left justified, and padded with blanks.

Versions of the DBRC API macro

Because parameter lists can change from one release of IMS to the next, the functions provided by the DSPAPI macro have a version associated with them. You must specify that version number or a later version to use the functions or parameters that are associated with a version.

If parameters have a version dependency, the parameter descriptions in each request type identify the minimum version number that is required.

The output block version number of the DBRC API is:

- 6.0 for IMS 14
- 5.0 for IMS Version 13
- 4.0 for IMS Version 12

The DBRC API token

The DBRC API token is a 4 byte field that is used to relate a series of API requests.

Your program receives this token when a DSPAPI FUNC=STARTDBRC macro is issued. This token must be supplied with all other macro calls that are associated with this instance of the STARTDBRC macro. The token is no longer valid after a DSPAPI FUNC=STOPDBRC macro call. Your program does not receive a token if the macro encounters a severe error (return code X'0000000C'). If the function receives a warning error (return code X'00000004'), find the meaning of the accompanying reason code in Chapter 34, “DBRC start request (STARTDBRC),” on page 439 to determine what action is needed.

Related concepts

“How an application program establishes the DBRC API environment” on page 342
Your application program establishes the application programming interface (API) environment by using the DSPAPI FUNC=STARTDBRC macro. This macro function initializes the API, creates a connection to the Database Recovery Control (DBRC) RECON data sets, and returns the API token.

Macro forms of the DSPAPI macro

There are four different macro forms for the DSPAPI macro: Standard (S), List (L), Modify (M), and Execute (E), with two variations, COMPLETE and NOCHECK. The List, Modify, and Execute forms are usually used in combinations when writing reentrant programs or when the application issues multiple requests.

Defaults are taken where necessary and less validity checking is performed than for the Standard Form. The following are explanations about when and why to use each form.

Standard form (default)

Use the standard form of the macro (MF=S or MF=S,*list*) to generate and modify an inline parameter list. If your program is reentrant, do not use the standard form of the macro because reentrant code cannot be modified. With few exceptions, if you use the standard form of the macro in writing reentrant code, the execution of the code results in an abend. The standard form of the macro serves three functions:

- Creates an inline parameter list
- Modifies the parameter list with the parameters specified on the request

- Sends the request to the API

Using the standard form, you can optionally assign a label to the generated parameter list by specifying MF=(S,*list*) where *list* specifies the name of the label assigned to the parameter list created by this form of the macro. The standard MF=S form of the macro is the default.

List form

Use the list form of the macro (MF=L,*list*) to generate a labeled, inline parameter list. This list is populated with the parameter values specified on the macro. The list form does not modify an existing list and does not send a request to the API. In effect, the list form creates a template that can be used as the target of a modify form or an execute form (the real list you plan to use). If the parameter list is generated in reentrant code, it cannot be modified. Therefore, you must obtain enough storage to hold the parameter list and use this storage as the target of the modify or list form.

list specifies the name of the label assigned to the parameter list created by this form of the macro.

Register notation is not compatible with the List Form of the macro. Instead, an ADCON of zero is generated.

Modify form

Use the modify form of the macro (MF=M,*list*,COMPLETE | NOCHECK) to change the values specified on the macro in the parameter list specified by the list parameter. The modify form does not generate a parameter list and does not issue requests to DBRC.

list

A symbol or a general purpose register in the range 2 to 12 that specifies the address of the parameter list to be modified.

COMPLETE

Specifies that DBRC uses defaults (for the parameters that have defaults) for all parameters that are not in the list and performs validity checking for the parameters that are specified in the list.

NOCHECK

Specifies no defaults are set and the existing parameter list is used. Validity checking is minimal. However, invalid keyword combinations are flagged in error.

Execute form

Use the Execute form of the macro (MF=E,*list*,COMPLETE | NOCHECK) to:

- Modify a parameter list (generated by the list form) with new and additional allowable parameters you might not have specified on the List form
- Issue a request to the API

You can change the parameters on the macro with each subsequent invocation of the execute form of the macro.

list

Can be a symbol or a general purpose register in the range 2 to 12 and specifies the address of the parameter list to be used.

COMPLETE

Specifies that DBRC uses defaults (for the parameters that have defaults) for all parameters that are not in the list, and performs validity checking for the parameters that are specified in the list.

NOCHECK

Specifies no defaults are set and the existing parameter list is used. Validity checking is minimal. However, invalid keyword combinations are flagged in error.

The following syntax is a summary of the macro forms:

MF=S | L | M | E

Specifies the macro form:

MF=S | MF=(S,list)

Specifies the standard form of the macro. MF=S is the default.

MF=(L,list)
Specifies the list form of the macro.

MF=(M,list,COMPLETE |NOCHECK)
Specifies the modify form of the macro.

MF=(E,list,COMPLETE|NOCHECK)
Specifies the execute form of the macro.

Related reference

“Parameters for the AUTH request” on page 354
You can use this information to understand the parameters for the DBRC AUTH request. Each parameter is explained as it relates to the AUTH request syntax diagram.

“Parameters for the COMMAND request” on page 362
You can use this information to understand the parameters for the DBRC COMMAND request. Each parameter is explained as it relates to the COMMAND request syntax diagram.

“Backout query request (TYPE=BACKOUT)” on page 369
You can use the Backout query (DSPAPI FUNC=QUERY TYPE=BACKOUT) request to retrieve backout information from the RECON data set for a specific subsystem or all subsystems.

“Log query request (TYPE=LOG)” on page 407
You can use the Log query (DSPAPI FUNC=QUERY TYPE=LOG) request to retrieve log information from RECON for a specific instance of a subsystem.

“OLDS query request (TYPE=OLDS)” on page 416
You can use the OLDS query (DSPAPI FUNC=QUERY TYPE=OLDS) request to retrieve online log data set information from the RECON for a specific subsystem or all subsystems.

“RECON status query request (TYPE=RECON)” on page 427
You can use the RECON status query (DSPAPI FUNC=QUERY TYPE=RECON) request to retrieve information pertaining to the RECON data sets, including RECON header information as well as the status of each RECON data set.

“Subsystem query request (TYPE=SUBSYS)” on page 431
You can use the Subsystem query (DSPAPI FUNC=QUERY TYPE=SUBSYS) request to retrieve subsystem information from the RECON data set for a specific subsystem or all subsystems.

“DBRC release buffer request (RELBUF)” on page 437
You can use the DBRC release buffer request (DSPAPI FUNC=RELBUF) to release storage that was acquired as a result of DBRC query, command, authorization, and unauthorization requests.

“DBRC stop request (STOPDBRC)” on page 443
You can use the STOPDBRC request to terminate the DBRC application and to stop DBRC.

“DBRC start request (STARTDBRC)” on page 439
You can use the DBRC start request (STARTDBRC) to initialize the DBRC API and to start DBRC.

“DBRC unauthorization request (UNAUTH)” on page 445

You can use the UNAUTH request to explicitly remove authorization to a database or area. Authorization by an application is implicitly removed by the STOPDBRC request. UNAUTH is the opposite of FUNC=AUTH.

Query output block header

This example illustrates the general format of the output from a Query request. The sample DSECT that follows this figure describes in detail the fields of the storage blocks and their relationship to each other.

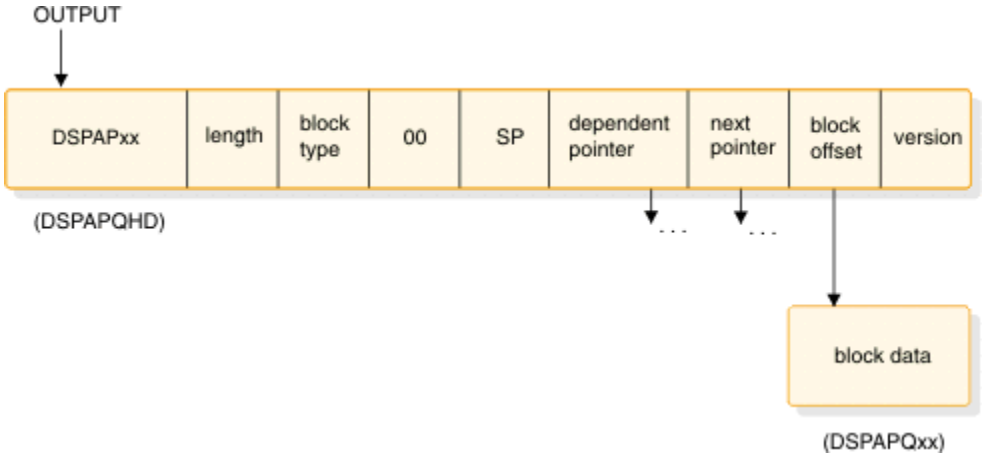


Figure 10. General format of the query output request

For STARTDBRC, SSID is the optionally specified subsystem ID. For STOPDBRC, SSID is the subsystem ID that is specified on the STARTDBRC request.

DSPAPQHD - QUERY output block header

```
  OFFSET   OFFSET
 DECIMAL      HEX          TYPE             LENGTH   NAME (DIM)          DESCRIPTION
========     ========      =========       ========   ==============
================================
       0       (0)         STRUCTURE          32   DSPAPQHD
       0       (0)         CHARACTER           8    APQHD_EYECATCHER   Output area eyecatcher
       8       (8)         SIGNED              4    APQHD_LENGTH       Block length, hdr + data
      12       (C)         SIGNED              2    APQHD_BLKTYPE      Block type
      14       (E)         UNSIGNED            2    *                  Reserved
      15       (F)    UNSIGNED            1    APQHD_SUBPOOL      Subpool ID
      16      (10)    ADDRESS             4    APQHD_DEPPTR       Ptr to block dependent
      20      (14)    ADDRESS             4    APQHD_NEXTPTR      Ptr to next block of the same type
      24      (18)    UNSIGNED            4    APQHD_BLKOFFSET    Offset to block data
      28      (1C)    SIGNED              4    APQHD_VERSION      Version of output block
```

Runtime considerations for the DBRC API

There are various considerations that you must address while the DBRC API is running such as DSPAPI and RECON access and how the API operates in an IMSplex.

DSPAPI macro access

In order to provide DBRC API (DSPAPI) macro access, your application must allocate IMS.SDFSRESL as either a JOBLIB or STEPLIB in your program's JCL because the DSPAPI is distributed with IMS.

RECON data set access

While the DBRC API is running, it uses up to three RECON data sets, which are allocated using the DD names RECON1, RECON2, and RECON3, or their respective alternate DD names. MDA members must be created for the data sets before the DBRC API can allocate RECONs.

The RECONs can be allocated:

- Dynamically by the API when the DSPAPI FUNC=STARTDBRC request is issued–this is the recommended method
- Dynamically by your application program
- Through your JCL

If the DBRC API allocates the RECONs, they are deallocated when the DSPAPI FUNC=STOPDBRC request is issued. Before the DBRC API can allocate the RECONs, MDA members must have been created for RECON1, RECON2, and RECON3. or their respective alternate DD names. The MDA members must exist either in a library allocated as //IMSDALIB, or in the //JOBLIB or //STEPLIB libraries. The DBRC API first searches in //IMSDALIB, if it exists, for the MDA members.

The IMS libraries concatenated to JOBLIB or STEPLIB are usually APF Authorized. If your program runs APF Authorized and the library containing the MDA members is not APF Authorized, allocate it using IMSDALIB as the DD name.

Requirement: All jobs accessing a set of RECON data sets must allocate the same data set by the same DD name. For example, RECON1=dsn1, RECON2=dsn2, and RECON3=dsn3. Failure to follow this convention causes serious damage to the RECONs.

The API uses only one set of RECONs between a FUNC=STARTDBRC request and its associated FUNC=STOPDBRC request. Your program, however, can use multiple sets of RECONs by deallocating the RECONs after the FUNC=STOPDBRC request and then allocating a new set of RECONs before issuing the next FUNC=STARTDBRC request. In order for this to work, your program must dynamically allocate the RECONs.

Related reference
DFSMDA macro (System Definition)

RECON access authority

You can set three levels of access control for DBRC. Each level provides different permissions to users that access the RECON data sets.

DBRC allows three levels of access control:

- Users who delete and define information in the RECON data sets require ALTER authority.
- Users who only update the RECON data sets require UPDATE authority.
- Users who do not update the RECON data sets, but who want to query information from the RECON data sets, require READ authority.

The READONLY keyword for the DBRC Start API request (STARTDBRC) allows a user to specify a READONLY option. READONLY is also a JCL EXEC PARM on the Database Recovery Control utility (DSPURX00). READONLY specifies that the job is not allowed to make any updates to the RECON data sets. READONLY is necessary if the user submitting the job does not have UPDATE (or higher) authority for the data sets. When READONLY is specified, there is no recovery action taken when an error occurs on a RECON data set. Instead of swapping in a spare data set, the job is terminated.

Time stamp format for DBRC requests

Time stamps that are associated with DBRC requests (either input or output) follow a packed decimal UTC time format. DBRC ignores the value of the offset field in time stamps that are provided on Query requests.

The time stamp is in the following format:

```
yyyydddFhhmmssthmijufqqs
```

Where:

yyyy
year (0000 to 9999)

ddd
day (000 to 366)

F
Hexadecimal character for padding purposes (X'F')

hh
hour (0 to 23)

mm
minute (0 to 59)

ss
second (0 to 59)

thmiju
millionths of a second (microseconds, 000000 to 999999)

fqqs
Time zone offset:

f
Flag bits, normally 0 for UTC representation

qq
Quarter hours (32/4=8)

s
Sign (D is negative, C is positive)

How DBRC uses the output data set

While running under the DBRC API, DBRC might output messages and other information to a data set defined by the DD name SYSPRINT, or by a DD name you specify. If your program already uses the DD name SYSPRINT, you can specify an alternate DD name for the API to use.

This output data set might be on tape, DASD, a printer, or routed through the output stream (SYSOUT). The data set might be allocated by your program through JCL or dynamically allocated prior to invoking the DBRC API. The attributes for this data set are RECFM=FBA, LRECL=121. Do not specify the block size because DBRC will use a system determined block size (the system determines an optimal blocksize for the device).

Wildcard support for name parameters for Query requests

For more flexible Query requests, you can use a wildcard (an asterisk) in several keyword parameters that specify names. This wildcard support enables you to specify a name pattern for your query, expanding some queries and filtering others.

The asterisk can be used in two formats:

- Use the asterisk alone to request information for all DB names, groups, or subsystems, depending on the query type.
- Use the asterisk at the end of a name to request information for DB names, groups, or subsystems whose names match the patterns. In this case, the asterisk must be preceded by at least one alphabetic character.

Use of wildcards is supported in the following parameters:

Table 82. Parameters for DBRC QUERY with wildcard support

Parameter name	Query type	Syntax
DBNAME	DB	DBNAME=*dbname* | *dbname**
GROUP	All xxxxGROUP	GROUP= * | *name* | *name**

Table 82. Parameters for DBRC QUERY with wildcard support (continued)

Parameter name	Query type	Syntax
NAME	All xxxxGROUP	NAME= * | *name* | *name** NAME=*
SSID	• BACKOUT • OLDS • SUBSYS	SSID= * | *symbol* | *symbol**

Chapter 29. DBRC API security features

You might want to limit access to the RECON data set to certain users. With the DBRC API, you can give installation control to individual DBRC API requests that users might issue.

The following table lists the DBRC API requests and the resource profiles used by the security product to protect each request. The symbol * indicates a wildcard value.

Table 83. DBRC API requests and resource profiles

Function	Type	Parameter	Resource
STARTDBRC or STOPDBRC	N/A	No parameter specified	hlq.STDBRC This resource is used if no ssid is specified.
		SSID=ssid	hlq.STDBRC.ssid For STARTDBRC, ssid is the optionally specified subsystem ID. For STOPDBRC, ssid is the subsystem ID specified on the STARTDBRC request.
RELBUF	N/A	N/A	N/A
QUERY	RECON	N/A	hlq.LIST.RECON
QUERY	DB	DBNAME=name	hlq.LIST.DB.name
		DBNAME=name*	hlq.LIST.DB.ALL
		DBLIST=dblist	hlq.LIST.DB.ALL
		LOC=FIRST | NEXT	hlq.LIST.DB.ALL
QUERY	PART	DBNAME=name	hlq.LIST.DB.name
		PARTNAME=name	hlq.LIST.DB.name
QUERY	DBDS	DBNAME=name	hlq.LIST.DBDS.name
		GROUP=grpname	hlq.LIST.DBDS.grpname
QUERY	LOG	STARTIME	hlq.LIST.LOG.STARTIME
		FROMTIME | TOTIME	hlq.LIST.LOG.ALL
QUERY	OLDS	SSID=ssid | ssid*| *	hlq.LIST.LOG.ALLOLDS
QUERY	SUBSYS	SSID=ssid	hlq.LIST.SUBSYS.ssid
		SSID=ssid*	hlq.LIST.SUBSYS.ALL
		SSID=*	hlq.LIST.SUBSYS.ALL
		SSTYPE=ALL	hlq.LIST.SUBSYS.ALL
		SSTYPE=BATCH	hlq.LIST.SUBSYS.BATCH
		SSTYPE=ONLINE	hlq.LIST.SUBSYS.ONLINE
		SSTYPE=DBRCAPI	hlq.LIST.SUBSYS.DBRCAPI

Table 83. DBRC API requests and resource profiles (continued)

Function	Type	Parameter	Resource
QUERY	BACKOUT	SSID=ssid	hlq.LIST.BKOUT.ssid
		SSID=ssid*	hlq.LIST. BKOUT.ALL
		SSID=*	hlq.LIST. BKOUT.ALL
QUERY	DBDSGROUP	GROUP=grpname	hlq.LIST.DBDSGRP.grpname
		GROUP=*|grpname*	hlq.LIST.DBDSGRP.ALL
QUERY	DBGROUP	GROUP=grpname	hlq.LIST.DBDSGRP.grpname
		GROUP=*|grpname*	hlq.LIST.DBDSGRP.ALL
QUERY	RECOVGROUP	GROUP=grpname	hlq.LIST.DBDSGRP.grpname
		GROUP=*|grpname*	hlq.LIST.DBDSGRP.ALL
QUERY	CAGROUP	GROUP=grpname	hlq.LIST.CAGRP.grpname
		GROUP=*|grpname*	hlq.LIST.CAGRP.ALL
QUERY	GSGROUP	GROUP=grpname	hlq.LIST.GSG.grpname
		GROUP=*|grpname*	hlq.LIST.GSG.ALL
COMMAND	N/A	N/A	Use current command resources.
AUTH UNAUTH	N/A	AUTHLIST=list	hlq.AUTH.dbname Each dbname in the list is verified.

Chapter 30. DBRC authorization request (AUTH)

You can use the DBRC authorization request to ensure that an invalid data sharing environment is not created. Database authorization is the process of requesting permission to access and use a database. In this context, the database is either a DL/I database or a Fast Path DEDB area.

The type of access requested can be:

- Exclusive (EX) control of the resource. Exclusive access is granted if the database is not currently authorized. After access is granted, any and all other authorization requests are not allowed. This type of authorization is used typically by a recovery utility.
- Read (RD) access to the database (also called read with integrity). Read access is granted if the database is not currently authorized as exclusive or update. After this access is granted, other authorization request for exclusive or update fails. This type of authorization is typically used by utilities that take non-concurrent (clean) image copies.
- Read-only (RO) access to the database (also called dirty read). Read-only access is granted unless the database is currently authorized as exclusive. Once granted, other authorization request for exclusive fail. Read-only access does not prevent the database from being updated while the application is reading it. This type of access is typically used by utilities that take concurrent image copies.

An application must register with DBRC when it initializes in order to use this function. The requested database must be registered with DBRC.

The database is unauthorized explicitly by the DSPAPI FUNC=UNAUTH macro, or implicitly by the DSPAPI FUNC=STOPDBRC macro.

You can specify one or more database names for the authorization and unauthorization functions. Databases do not have to be unauthorized in the same order that they are authorized. For instance, if multiple databases in a list are authorized, they can be unauthorized one at a time in any order. To change the type of access, unauthorize the database and then authorize the database again.

There are some situations in which standard database authorization is denied, such as when the "Prohibit Authorization" flag is on for a database, or if one or more of its DBDSs require an image copy. If the intent of the application is database image copy, recovery or reorganization, you can grant the authorization by using the UTILITY keyword on the FUNC=AUTH request.

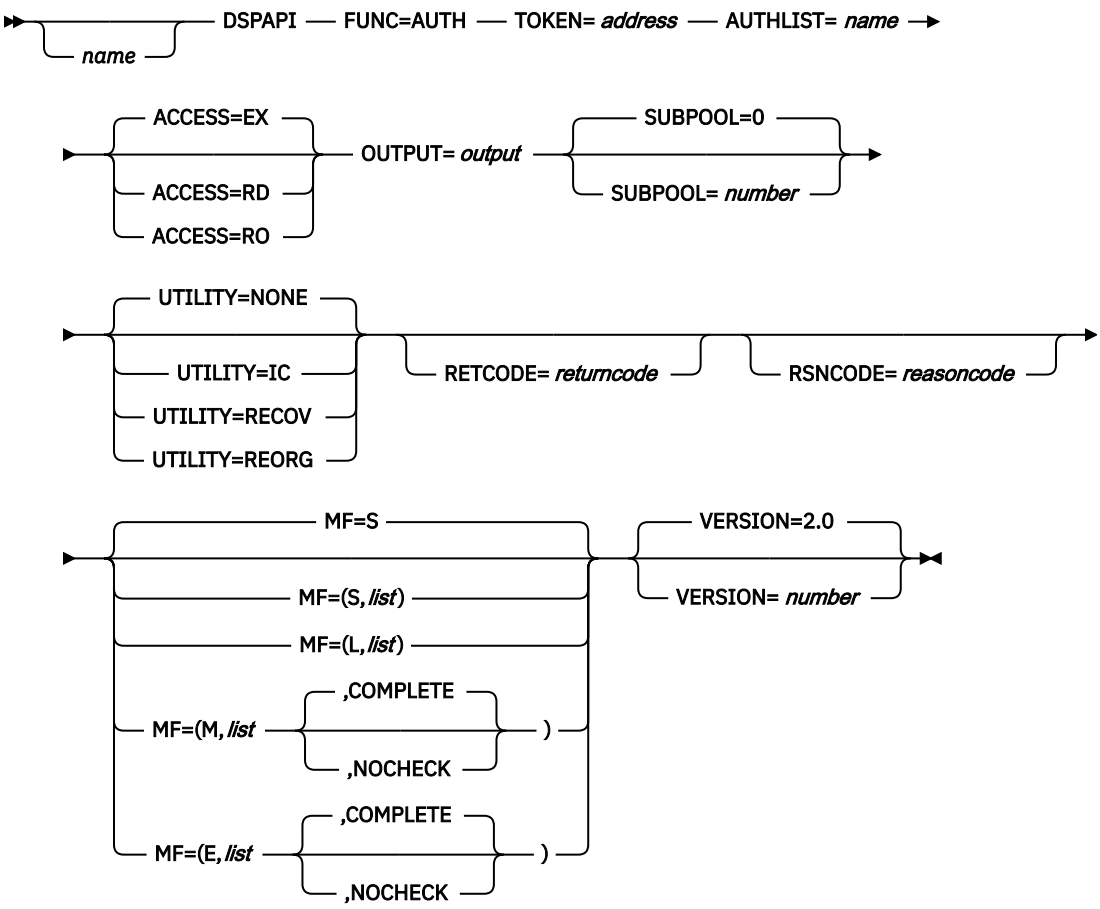
Related concepts

“DBRC API” on page 341
Your applications can obtain services from Database Recovery Control (DBRC) through the DBRC application programming interface (API), a release-independent, assembler macro interface. The

application obtains these services by issuing DBRC API requests to DBRC, and DBRC returns the results to an area in storage where the application can retrieve them.

Syntax for the AUTH request

You can use this syntax diagram to understand the format of the DBRC AUTH request.



Parameters for the AUTH request

You can use this information to understand the parameters for the DBRC AUTH request. Each parameter is explained as it relates to the AUTH request syntax diagram.

name *name*
Begin the name in column 1.

TOKEN=*address* | (2-12)
Specifies the address of the API token that was returned on the FUNC=STARTDBRC macro.

AUTHLIST=*name* | (2 - 12)
Specifies the list of database names or Fast Path areas to be authorized. The list consists of a fullword that contains the number of elements in the list, a fullword that contains the length of an element, followed by one or more elements. Each element consists of an 8-character DB name or Fast Path DEDB name and 8 characters of blanks (X'40') or a Fast Path area name.

ACCESS=EX | RD | RO
Specifies exclusive (EX), read (RD) or read-only (RO) authorization is requested. The default is EX.

OUTPUT=*output* | (2 - 12)
Specifies a field to receive a pointer to the authorization output block DSPAPAUB.

The output address is 0 if no output was built. This can happen if nothing in the RECON data set satisfies the request or if an error occurs before any output could be built.

The storage for the output blocks is not preallocated by the caller. DBRC acquires storage from the specified subpool for these blocks. The caller must free this storage using the Buffer Release service (DSPAPI FUNC=RELBUF) specifying the returned output address.

SUBPOOL= 0 | *number*
Specifies the subpool number for the storage being obtained. If not specified, the default is the subpool specified by the FUNC=STARTDBRC request. Otherwise, subpool 0 is the default.

UTILITY=IC | RECOV | REORG | NONE
Specifies the intended utility function of the application. The functions include image copy (IC), database recovery (RECOV), database reorganization (REORG), or NONE. The default is NONE, indicating a normal database authorization request.

RETCODE=*return code* | (2-12)
If specified as a symbol, specifies the label of a word of storage to receive the return code. If specified as a register, the register must contain the address of a word of storage to receive the return code. Regardless of whether RETCODE is specified, register 15 contains the return code.

RSNCODE=*reason code* | (2-12)
If specified as a symbol, the symbol must be the label of a word of storage to receive the reason code. If specified as a register, the register must contain the address of a word of storage to receive the reason code. Regardless of whether RSNCODE is specified, register 0 contains the reason code.

MF=S | L | M | E
Specifies the macro form of the request.

VERSION=2.0 | *number*
Specifies the version number of the parameter list to be generated by this macro.

To use the parameters associated with a version, you must specify the number of that version or a later version. If you specify an earlier version level, the parameter is not accepted by macro processing, and an error message is issued at assembly time. If parameters have a version dependency, the parameter descriptions with each request type identify the version number required.

The valid version number for the FUNC=AUTH request is 2.0 (the default).

Related concepts

“Macro forms of the DSPAPI macro” on page 344
There are four different macro forms for the DSPAPI macro: Standard (S), List (L), Modify (M), and Execute (E), with two variations, COMPLETE and NOCHECK. The List, Modify, and Execute forms are usually used in combinations when writing reentrant programs or when the application issues multiple requests.

Related reference

“AUTH output block” on page 359
This example contains the output block that is returned by the AUTH request. The output block contains an array of authorized databases and indicates if the AUTH request was successful.

Return and reason codes for AUTH

You can use this table to search for reason and return codes for the DBRC AUTH request. Each code is accompanied by the code type and an explanation of the code.

Reason and return codes for the AUTH request

Table 84. DSPAPI FUNC=AUTH return and reason codes

Code type	Return code	Reason code	Meaning
	X'00000000'	X'00000000'	Request completed successfully.

Table 84. DSPAPI FUNC=AUTH return and reason codes (continued)

Code type	Return code	Reason code	Meaning
Warning	X'00000008'	X'C1000001'	One or more entries in the AUTHLIST can not be processed. A reason code has to be set in the corresponding entry in the AUTH output block.
Severe error - No AUTH block returned	X'0000000C'	X'C1000001'	Application is not signed on to DBRC.
	X'0000000C'	X'C1000002'	DBRC AUTH processing cannot complete because the application is not signed, and no subsystem record is found. This should not occur under normal conditions because an earlier check indicated that the subsystem was signed on.
	X'0000000C'	X'C9000001'	Invalid TOKEN. The TOKEN block passed to the API is not recognized as a TOKEN created by a FUNC=STARTDBRC call.
	X'0000000C'	X'C900000A'	The TCB address is not the same as the TCB address under which the STARTDBRC service was issued.
Storage error	X'00000028'	X'C1000001'	Error obtaining storage for the AUTH output block.
Internal error	X'0000002C'	X'C1000001'	Error attempting to start RECON multiple update processing.
	X'0000002C'	X'C1000002'	Error attempting to end RECON multiple update processing.
	X'0000002C'	X'C1000003'	Entry in AUTH output block could not be found. This should not occur.
	X'0000002C'	X'C1000004'	Internal error encountered during DBRC authorization processing.
	X'0000002C'	X'C1000005'	Internal error encountered during DBRC authorization processing - invalid parameters.
Parameter error	X'00000030'	X'C1000001'	No AUTHLIST passed.
	X'00000030'	X'C1000002'	AUTHLIST passed with no entries
	X'00000030'	X'C1000003'	Duplicate elements in AUTHLIST.
	X'00000030'	X'C1000004'	Missing or invalid OUTPUT parameter.
	X'00000030'	X'C9000001'	The function (FUNC) specified in the parameter list passed to the API is invalid.
	X'00000030'	X'C9000002'	Invalid TOKEN address. The address of the field containing the API TOKEN failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000003'	Invalid RETCODE address. The address of the field containing the API RETCODE failed validity checking. The address specifies storage not owned by the calling program.

Table 84. DSPAPI FUNC=AUTH return and reason codes (continued)

Code type	Return code	Reason code	Meaning
	X'00000030'	X'C9000004'	Invalid RSNCODE address. The address of the field containing the API RSNCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000005'	Invalid OUTPUT address. The address of the field containing the API OUTPUT failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C900000A'	An incorrect VERSION value was specified for the requested function (FUNC).
	X'00000030'	X'C900001A'	Invalid AUTHLIST address. The address of the field containing the API AUTHLIST failed validity checking. The address specifies storage not owned by the calling program.

Related reference

"APAUB_RsnCode for AUTH output block" on page 357
You can use this table to search for APAUB_RsnCode values for AUTH request return and reason codes. Each code is accompanied by an explanation of the code.

DBRC request return codes (Messages and Codes)

APAUB_RsnCode for AUTH output block

You can use this table to search for APAUB_RsnCode values for AUTH request return and reason codes. Each code is accompanied by an explanation of the code.

When an AUTH output block (DSPAPAUB) is returned, one of the following reason codes is set in field APAUB_RsnCode for each element in the list of DBs or Areas in the request.

Table 85. APAUB_RsnCode values for AUTH request return and reason codes

APAUB_RsnCode	Meaning
X'00000000'	Request completed successfully.
X'C1000100'	Security error. SAF or the DBRC command authorization exit (DSPDCAX0) has determined that the user is not authorized to perform the request for this database or area.
X'C1000201'	The requested state and the current authorization state are incompatible. The database is authorized by another active or abnormally terminated IMS subsystem, and its authorization state is incompatible with the current authorization request.
X'C1000202'	The database is not registered in the RECON data set. You may also have set up your AUTHLIST incorrectly. For Fast Path, you must specify the DEDB name and the area name. For non-Fast Path the element is an eight-character DB name and eight characters of blanks.
X'C1000203'	The database is marked as prohibiting further authorization for one of the following reasons: a global **/DBR**, a global **/STOP**, an **UPDATE DB STOP**, or a **CHANGE.DB(NOAUTH)** command.
X'C1000204'	The database is authorized only if the requested state is "READ" or "READ-GO" because of a global **/DBDUMP** or an **UPDATE DB STOP(UPDATES)** command.

Table 85. APAUB_RsnCode values for AUTH request return and reason codes (continued)

APAUB_RsnCode	Meaning
X'C1000205'	The database is marked as needing an image copy.
X'C1000206'	The database is marked as needing recovery.
X'C1000207'	The database is marked as needing backout.
X'C100020A'	The database has been previously authorized to the subsystem
X'C100020B'	An invalid parameter is found during the DB usage compatibility evaluation process. Possibly, the database record in the RECON data set is bad.
X'C100020C'	The current authorization state in DBRC is invalid because of an unauthorization error.
X'C100020D'	An error occurred in DBRC while trying to process an authorization request.
X'C100020F'	The database is already authorized to an active IMS subsystem. All subsystems must be abnormally terminated.
X'C1000214'	DB authorization failed because the DB was not registered with DBRC and the RECON FORCER option is in effect, which requires that all databases must be registered.
X'C1000220'	The HALDB needs to be initialized.
X'C1000221'	An attempt has been made to authorize the HALDB master. Authorization can only be requested at the partition level.
X'C1000223'	The DB partition cannot be authorized until a high key is defined. A key is required because the HALDB master does not use a Partition Selection Routine.
X'C1000224'	Image copy not allowed during HALDB OLR processing.
X'C1000225'	Loading into an M-V DBDS of a partition database is not allowed
X'C1000226'	Offline reorganization is not allowed when HALDB OLR is active and the HALDB OLR is owned by an IMS subsystem.
X'C1000228'	Database is being reorganized.
X'C1000229'	Batch update, recovery utility, and reorganization utility are not allowed when DB Quiesce is in progress.
X'C100022A'	Image copy utility is not allowed when DB Quiesce is in progress and DB Quiesce Held state is not yet achieved.
X'C10003xx'	An authorization reason code was received that should not apply to a DBRC application. xx is the hexadecimal equivalent to the FxFx reason returned. This reason code is useful for diagnostics.

AUTH output block mapping

You can use this figure to understand the format of the output from a TYPE=AUTH request. The output block for the TYPE=AUTH request begins with a standard header that is mapped by the DSPAPQHD. The data portion of this output block is mapped by DSPAPAUB.

The following figure illustrates the format of the output from a TYPE=AUTH.

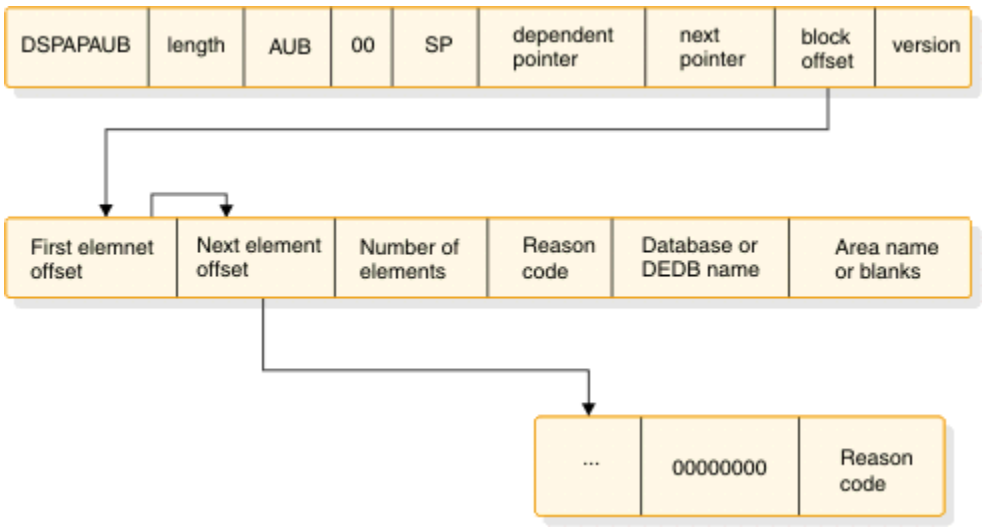


Figure 11. Format for a TYPE=AUTH output

AUTH output block

This example contains the output block that is returned by the AUTH request. The output block contains an array of authorized databases and indicates if the AUTH request was successful.

Example of output block mapped by the DSPAPAUB

```
================================================================================
DSPAPAUB

  OFFSET   OFFSET
 DECIMAL      HEX  TYPE        LENGTH  NAME (DIM)     DESCRIPTION
======== ======== ========= ========  ============== ==========================
       0      (0) STRUCTURE        8  DSPAPAUB       AUTH/UNAUTH block
       0      (0) UNSIGNED         4   APAUB_OFFSET  Offset to first element
       4      (4) SIGNED           4   APAUB_ELCOUNT Number of elements in list

  OFFSET   OFFSET
 DECIMAL      HEX  TYPE        LENGTH  NAME (DIM)     DESCRIPTION
======== ======== ========= ========  ============== ==========================
       0      (0) STRUCTURE       24  APAUB_ELEMENT
       0      (0) UNSIGNED         4   APAUB_OFFNEXT Offset to next element
       4      (4) SIGNED           4   APAUB_RSNCODE Reason code
       8      (8) CHARACTER        8   APAUB_DBNAME  Database or DEDB name
      16     (10) CHARACTER        8   APAUB_AREANAME
                                                     Area name or blanks

CONSTANTS

  LEN  TYPE           VALUE          NAME            DESCRIPTION
=====  =========  ================  ============== ==========================
    8  CHARACTER   DSPAPAUB         APAUB_EYECATCHER
```

Related reference

"Parameters for the AUTH request" on page 354
You can use this information to understand the parameters for the DBRC AUTH request. Each parameter is explained as it relates to the AUTH request syntax diagram.

Chapter 31. DBRC command request (COMMAND)

You can use the DSPAPI FUNC=COMMAND request to issue a DBRC utility command from your application program. All DBRC commands are accepted on this request, except for the LIST command.

All output generated by the DBRC command request is returned in an API output block rather than sent to SYSPRINT. If DBRC command authorization is enabled, commands entered through the API are produce the same results as the commands executed through the DBRC utility. An IMS DD statement might be required. If GENJCL commands are executed, both a JCLPDS DD statement (or the DD name you supply with the JCLPDS parameter) and a JCLOUT DD statement (or the DD name you supply with the JCLOUT parameter) are required.

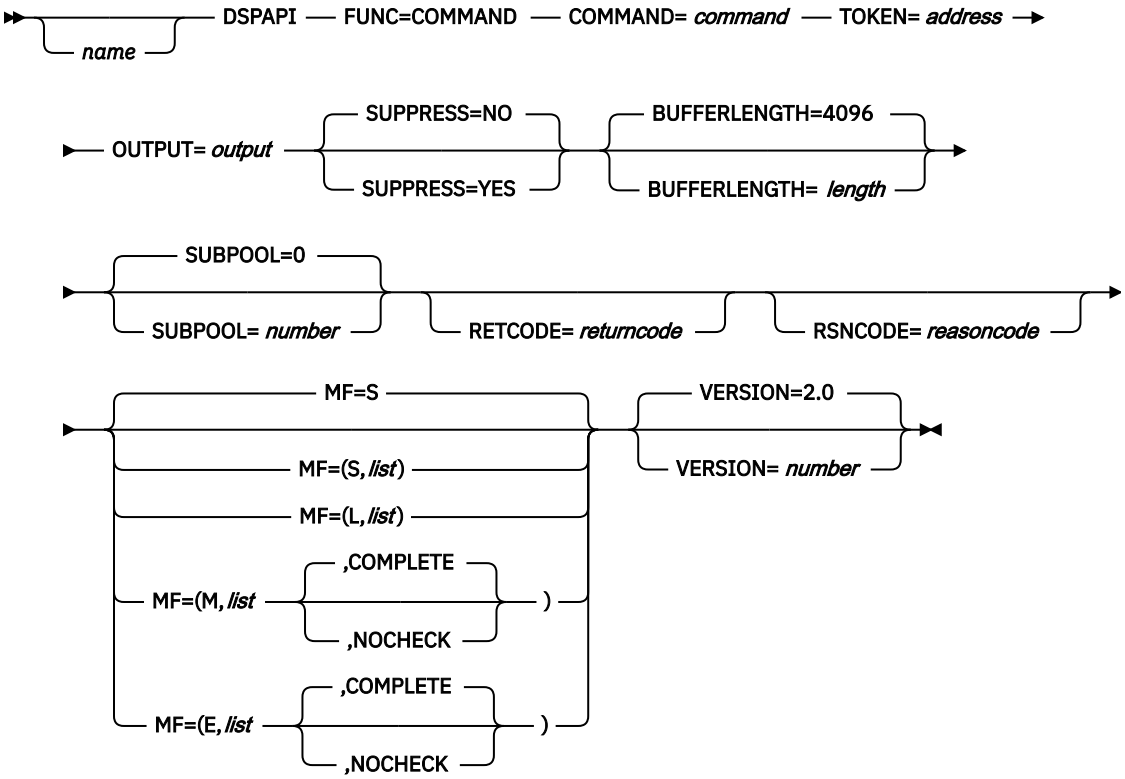
Related concepts

"DBRC API" on page 341

Your applications can obtain services from Database Recovery Control (DBRC) through the DBRC application programming interface (API), a release-independent, assembler macro interface. The application obtains these services by issuing DBRC API requests to DBRC, and DBRC returns the results to an area in storage where the application can retrieve them.

Syntax for the COMMAND request

You can use this syntax diagram to understand the format of the Database Recovery Control (DBRC) COMMAND request.



Parameters for the COMMAND request

You can use this information to understand the parameters for the DBRC COMMAND request. Each parameter is explained as it relates to the COMMAND request syntax diagram.

name
Specifies the name parameter. If used, begins in column 1.

COMMAND=*symbol* | (2 - 12)
Specifies the address of a DBRC utility command to execute. The command consists of a header followed by a DBRC command. The header is a full word that contains the length (in bytes) of the following command. The DBRC command must conform to DBRC command syntax, except that the command cannot be continued. Separators, which are blanks, a comma or a comment, are allowed anywhere in the command where a separator is needed and may validly appear at the beginning of the command.

TOKEN=*symbol* | (2 - 12)
Specifies the address of a 4-byte field to receive the API token. This token must be included in all subsequent requests that are associated with this DSPAPI request.

OUTPUT=*output* | (2 - 12)
Specifies a field to receive a pointer to the block that contains the information for the command.

SUPPRESS= | NO | YES
Specifies whether or not the command output is to be returned. SUPPRESS=YES indicates that command output is to be returned only if the command does not complete successfully (return code is not zero). SUPPRESS=NO indicates that command output is always returned. SUPPRESS=NO is the default.

BUFFERLENGTH=4096 | *number* | (2 - 12)
Specifies the length of a buffer to receive the output that is generated by executing the command. If a register is specified, the register must contain the desired length. The maximum length allowed is 32 760. If necessary, the length is rounded up to a double-word boundary. The default and minimum value is 4096.

SUBPOOL=0 | *number*
Specifies the subpool number for the storage being obtained. If not specified, the default is the subpool specified by the FUNC=STARTDBRC request. Otherwise, subpool 0 is the default.

RETCODE=*return code* | (2-12)
If specified as a symbol, specifies the label of a word of storage to receive the return code. If specified as a register, the register must contain the address of a word of storage to receive the return code. Regardless of whether RETCODE is specified, register 15 contains the return code.

RSNCODE=*reason code* | (2-12)
If specified as a symbol, the symbol must be the label of a word of storage to receive the reason code. If specified as a register, the register must contain the address of a word of storage to receive the reason code. Regardless of whether RSNCODE is specified, register 0 contains the reason code.

MF=S | L | M | E
Specifies the macro form of the request.

VERSION=2.0 | *number*
Specifies the version number of the parameter list to be generated by this request.

To use the parameters associated with a version, you must specify the number of that version or a later version. If you specify an earlier version level, the parameter is not accepted for processing and an error message is issued at assembly time. If parameters have a version dependency, the parameter descriptions with each request type identify the version number required.

The default version is 2.0.

Related concepts

"Macro forms of the DSPAPI macro" on page 344

There are four different macro forms for the DSPAPI macro: Standard (S), List (L), Modify (M), and Execute (E), with two variations, COMPLETE and NOCHECK. The List, Modify, and Execute forms are usually used in combinations when writing reentrant programs or when the application issues multiple requests.

Related reference

DBRC command syntax (Commands)

Introduction to the DBRC commands (Commands)

Return and reason codes for the COMMAND request

You can use this table to search for reason and return codes for the DBRC COMMAND request. Each code is accompanied by the code type and an explanation of the code.

Table 86. DSPAPI FUNC=COMMAND return and reason codes

Code type	Return code	Reason code	Meaning
	X'00000000'	X'00000000'	Request completed successfully. The DBRC command completed with return code 0.
Partial success	X'00000004'	X'C3000001'	The DBRC command completed with a non-zero return code.
Severe error	X'0000000C'	X'C3000001'	The LIST command is not allowed. Use the Query function.
	X'0000000C'	X'C9000001'	Invalid TOKEN. The TOKEN block passed is not recognized as a TOKEN created by a FUNC=STARTDBRC call.
	X'0000000C'	X'C900000A'	The TCB address is not the same as the TCB address under which the STARTDBRC service was issued.
Storage error	X'00000028'	X'C3000001'	Error obtaining storage for a Command block.
Parameter error	X'00000030'	X'C3000001'	Missing or invalid COMMAND parameter.
	X'00000030'	X'C3000002'	Invalid command length. Must be greater than zero.
	X'00000030'	X'C3000003'	Missing or invalid OUTPUT parameter
	X'00000030'	X'C3000004'	Invalid BUFFERLENGTH value. Must be >= 0 and <= 32760.
	X'00000030'	X'C9000001'	The function (FUNC) specified in the parameter list is invalid.
	X'00000030'	X'C9000002'	Invalid TOKEN address. The address of the field containing the API TOKEN failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000003'	Invalid RETCODE address. The address of the field containing the API RETCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000004'	Invalid RSNCODE address. The address of the field containing the API RSNCODE failed validity checking. The address specifies storage not owned by the calling program.

Table 86. DSPAPI FUNC=COMMAND return and reason codes (continued)

Code type	Return code	Reason code	Meaning
	X'00000030'	X'C9000005'	Invalid OUTPUT address. The address of the field containing the API OUTPUT failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C900000A'	An incorrect VERSION value was specified for the requested function (FUNC).
	X'00000030'	X'C9000019'	Invalid COMMAND address. The address of the field containing the API COMMAND failed validity checking. The address specifies storage not owned by the calling program.

COMMAND output block mapping

These examples illustrate output block mapping for the DBRC command request. You can use the examples to understand how the DSPAPCMD output block header is structured and how the output appears.

The following figure illustrates command output block mapping.

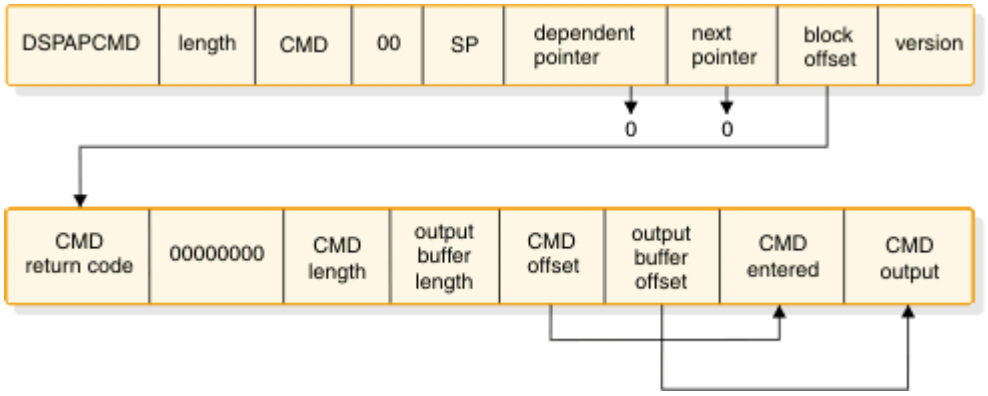


Figure 12. Mapping of command output block

Example of storage block mapped by the DSPAPCMD

Each storage block begins with a standard header that is mapped by the DSPAPQHD. The data portion of this output block is mapped by DSPAPCMD, as illustrated in the following example.

```
DSPAPCMD

  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)     DESCRIPTION
======== ======== ========= ========  ============== ============================
       0      (0) STRUCTURE       24  DSPAPCMD
       0      (0) SIGNED           4   APCMD_RETCODE DBRC command return code
       4      (4) SIGNED           4   *             Reserved
       8      (8) SIGNED           4   APCMD_CMDLEN  Length of command entered
      12      (C) SIGNED           4   APCMD_BUFFLEN Length of command output
      16     (10) UNSIGNED         4   APCMD_CMDOFF  Offset to command buffer
      20     (14) UNSIGNED         4   APCMD_BUFFOFF Offset to command output

 OFFSET   OFFSET
DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)     DESCRIPTION
======= ======== ========= ========  ============== ============================
      0      (0) STRUCTURE        *  APCMD_COMMAND  Command as entered
```

Example of DSPAPCMD output block header

In the following example, the DSPAPCMD output block header contains the status of the request, returns an echo of the command, and any command output. The following example describes the DSECT for the DBRC command output.

```
    Each output line is mapped by the following structure
================================================================================

  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)                    DESCRIPTION
======== ======== ========= ========  ==============                ==================
       0      (0) STRUCTURE         5  APCMD_OUTPUT_LINES
       0      (0) SIGNED            2  APCMD_OUTPUT_LINELEN   Length of line
       2      (2) SIGNED            2   *                         Reserved
       4      (4) CHAR VARY         1  APCMD_OUTPUT_DATA      Output data


CONSTANTS

  LEN  TYPE           VALUE          NAME                           DESCRIPTION
=====  =========  ================  ===============                ==================
    8  CHARACTER  DSPAPCMD          APCMD_EYECATCHER
```

Related reference

“DBDS query request (TYPE=DBDS)” on page 394
You can use the DSPAPI FUNC=QUERY TYPE=DBDS request to retrieve information from the RECON data set for one or more DBDSs in a non-HALDB database, a HALDB partition, a DBDS group, or a CA group. You can also request recovery related information for the data set, including allocation, image copy, recovery, and reorganization information.

Chapter 32. DBRC query request (QUERY)

You can use the DBRC Query request (DSPAPI FUNC=QUERY) along with the TYPE parameter to retrieve various types of information from the RECON data set.

- Backout (TYPE=BACKOUT)
- Database (TYPE=DB) - This variation of QUERY returns database registration and status information for:
  - Full-function databases
  - Fast Path databases
  - HALDB databases
  - DBDS or area information and supporting recovery-related information for each DBDS or area (allocation, image copy, recovery, and reorganization)
- Database partitioning (TYPE=PART)
- DBDS or area information (TYPE=DBDS)
- Group and member information for the following group types:
  - Change Accumulation (TYPE=CAGROUP). CA execution information can also be returned.
  - DBDS (TYPE=DBDSGROUP)
  - Database (TYPE=DBGROUP)
  - Recovery (TYPE=RECOVGROUP)
- Log, Recovery and System Log Data Set (TYPE=LOG)
- Online Log Data Set (TYPE=OLDS)
- RECON status (TYPE=RECON) - This variation of QUERY returns RECON header information, as well as the status of the RECON configuration.
- Subsystem (TYPE=SUBSYS)

Output from query requests are time consistent and access to the RECON data set is restricted during the processing of the request.

If you enable parallel RECON access, then the Query API request returns output as if you specified LIST.xxx STATIC (the RECON data set is not accessible while DBRC processes the Query request).

Related concepts

"DBRC API" on page 341
Your applications can obtain services from Database Recovery Control (DBRC) through the DBRC application programming interface (API), a release-independent, assembler macro interface. The application obtains these services by issuing DBRC API requests to DBRC, and DBRC returns the results to an area in storage where the application can retrieve them.

Related reference

"Backout query request (TYPE=BACKOUT)" on page 369
You can use the Backout query (DSPAPI FUNC=QUERY TYPE=BACKOUT) request to retrieve backout information from the RECON data set for a specific subsystem or all subsystems.

"Database query request (TYPE=DB)" on page 373
You can use the Database Query request (DSPAPI FUNC=QUERY TYPE=DB) to retrieve information from the RECON concerning one or more registered databases.

"HALDB partition query request (TYPE=PART)" on page 420
You can use the DSPAPI FUNC=QUERY TYPE=PART request to retrieve information for a particular HALDB partition from the RECON data set. You can request data set information for a specific DBDS or all DBDSs in the partition, and can optionally request recovery-related information for the data set, including allocation, image copy, recovery, and reorganization information.

"DBDS query request (TYPE=DBDS)" on page 394

You can use the DSPAPI FUNC=QUERY TYPE=DBDS request to retrieve information from the RECON data set for one or more DBDSs in a non-HALDB database, a HALDB partition, a DBDS group, or a CA group. You can also request recovery related information for the data set, including allocation, image copy, recovery, and reorganization information.

"Group query request (TYPE=*GROUP)" on page 400
You can use the Group query (DSPAPI FUNC=QUERY TYPE=*GROUP) request to retrieve group and member information for various types of groups that are registered in the RECON data set.

"Log query request (TYPE=LOG)" on page 407
You can use the Log query (DSPAPI FUNC=QUERY TYPE=LOG) request to retrieve log information from RECON for a specific instance of a subsystem.

"OLDS query request (TYPE=OLDS)" on page 416
You can use the OLDS query (DSPAPI FUNC=QUERY TYPE=OLDS) request to retrieve online log data set information from the RECON for a specific subsystem or all subsystems.

"RECON status query request (TYPE=RECON)" on page 427
You can use the RECON status query (DSPAPI FUNC=QUERY TYPE=RECON) request to retrieve information pertaining to the RECON data sets, including RECON header information as well as the status of each RECON data set.

"Subsystem query request (TYPE=SUBSYS)" on page 431
You can use the Subsystem query (DSPAPI FUNC=QUERY TYPE=SUBSYS) request to retrieve subsystem information from the RECON data set for a specific subsystem or all subsystems.

Output from query requests

Requested information is returned to the calling application in a chain of one or more blocks in storage. The pointer to the beginning of this chain is returned in the area specified by the OUTPUT parameter of the Query request.

The storage for these blocks is not preallocated by the calling application. DBRC will acquire private storage for these blocks. It is the responsibility of the calling application to free this storage using the Buffer Release request (DSPAPI FUNC=RELBUF).

Each storage block begins with a standard header mapped by the DSPAPQHD.

Related concepts

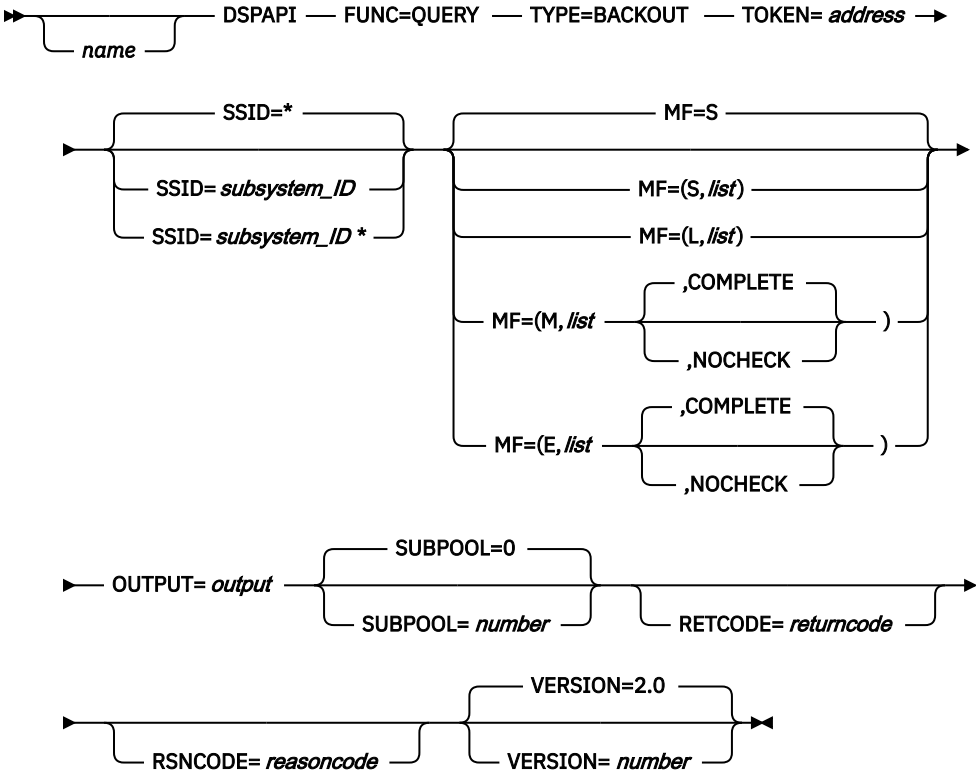
"DBRC API" on page 341
Your applications can obtain services from Database Recovery Control (DBRC) through the DBRC application programming interface (API), a release-independent, assembler macro interface. The

application obtains these services by issuing DBRC API requests to DBRC, and DBRC returns the results to an area in storage where the application can retrieve them.

Backout query request (TYPE=BACKOUT)

You can use the Backout query (DSPAPI FUNC=QUERY TYPE=BACKOUT) request to retrieve backout information from the RECON data set for a specific subsystem or all subsystems.

Syntax for the TYPE=BACKOUT query request



Parameters for the TYPE=BACKOUT query request

name
If used, begins in column 1.

TYPE=BACKOUT
Specifies that backout information is requested.

TOKEN=*symbol* | (2 - 12)
Specifies the address of a 4-byte field that was returned on the FUNC=STARTDBRC request.

SSID=* | *symbol* | *symbol** | (2 - 12)**
Specifies the subsystem name for the backout being queried. You can use the wildcard keyword * (an asterisk) alone to request information about all groups (SSID=*, which is the default). You can also add it at the end of a name to query all subsystems whose names match the pattern. In this case, the asterisk must be preceded by at least one alphabetic character.

MF=S | L | M | E
Specifies the macro form of the request.

OUTPUT=*output* | (2 - 12)
Required parameter that specifies a field to receive a pointer to the first block of backout information blocks.

The output address is 0 if no output was built. This can occur if nothing in the RECON satisfies the request or if an error occurs before any output could be built.

The storage for the output blocks is not pre-allocated by the caller. DBRC acquires storage from the specified subpool for these blocks. The caller must free this storage using the Buffer Release service (DSPAPI FUNC=RELBUF) and specify the returned output address.

SUBPOOL= 0 | *number*
Specifies the subpool number for the storage being obtained. If not specified, the default is the subpool specified by the FUNC=STARTDBRC request. Otherwise, subpool 0 is the default.

RETCODE=*returncode* | (2-12)
If specified as a symbol, specifies the label of a word of storage to receive the return code. If specified as a register, the register must contain the address of a word of storage to receive the return code. Regardless of whether RETCODE is specified, register 15 contains the return code.

RSNCODE=*reasoncode* | (2-12)
If specified as a symbol, the symbol must be the label of a word of storage to receive the reason code. If specified as a register, the register must contain the address of a word of storage to receive the reason code. Regardless of whether RSNCODE is specified, register 0 contains the reason code.

VERSION=2.0 | *number*
Specifies the version number of the parameter list that is generated by this macro.

To use the parameters associated with a version, you must specify the number of that version or a later version. If you specify an earlier version level, the parameter is not accepted by macro processing, and an error message is issued at assembly time. If parameters have a version dependency, the parameter descriptions with each request type identify the version number required.

Valid version numbers for the FUNC=QUERY TYPE=BACKOUT request are 1.0 and 2.0.

Return and reason codes for the TYPE=BACKOUT query request

Table 87. Return and reason codes for TYPE=BACKOUT query requests

Code type	Return codes	Reason codes	Meaning
	X'00000000'	X'00000000'	Request completed successfully.
Warning	X'00000008'	X'D87000001'	No backout records exist.
Severe error	X'0000000C'	X'C9000001'	Invalid TOKEN. The TOKEN block passed to the API is not recognized as a TOKEN created by a FUNC=STARTDBRC call.
	X'0000000C'	X'C900000A'	The TCB address is not the same as the TCB address under which the STARTDBRC service was issued.
	X'0000000C'	X'D8700100'	Security error. SAF or the DBRC Command Authorization exit routine (DSPDCAX0) determined that the user is not authorized to perform the request.
Storage error	X'00000028'	X'D87000001'	Error obtaining storage for BACKOUT block.
	X'00000028'	X'D91000001'	An error occurred processing the request. DBRC will release storage that was obtained up to this point. However, another error was encountered during the attempt to release storage.
Internal error	X'0000002C'	X'D8000001'	RECON open failure.
	X'0000002C'	X'D8700001'	Failure locating the first or the specified backout record.
	X'0000002C'	X'D8700002'	Failure locating the next backout record.
Parameter error	X'00000030'	X'C9000001'	The function (FUNC) specified in the parameter list passed to the API is invalid.

Table 87. Return and reason codes for TYPE=BACKOUT query requests (continued)

Code type	Return codes	Reason codes	Meaning
	X'00000030'	X'C9000002'	Invalid TOKEN field address. The address of the field containing the API TOKEN failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000003'	Invalid RETCODE field address. The address of the field to contain the API RETCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000004'	Invalid RSNCODE field address. The address of the field to contain the API RSNCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000005'	Invalid OUTPUT field address. The address of the field to contain the OUTPUT address failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000010'	Invalid SSID field address. The address of the field containing the SSID failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'D80000001'	Missing or invalid OUTPUT parameter.
	X'00000030'	X'D80000002'	Invalid value specified for TYPE parameter.
	X'00000030'	X'D8700100'	When using a wildcard, at least one alphabetic character must precede the asterisk.
	X'00000030'	X'D8700101'	When using a wildcard, the asterisk must be the last character.

Output for TYPE=BACKOUT QUERY request

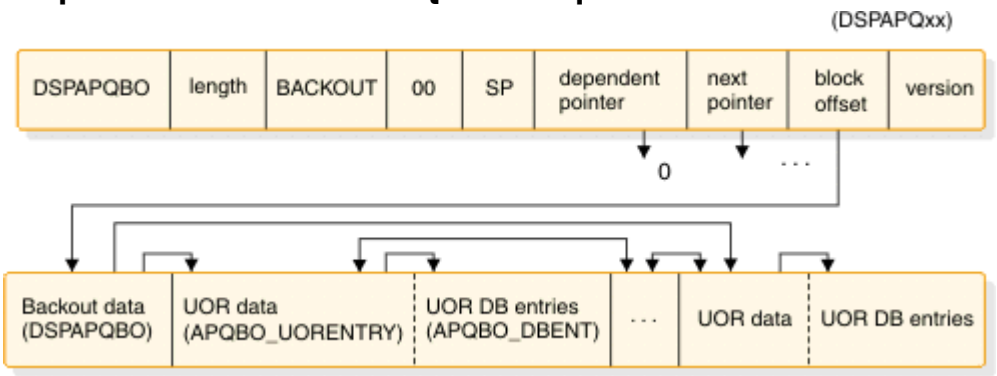


Figure 13. Format of the output from QUERY TYPE=BACKOUT output request

DSECT of DSPAPQBO

The following example is a sample DSECT describes in detail the fields of the storage blocks and their interrelationship.

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)       DESCRIPTION
======== ======== ========= ========  ==============   ============================
```

```
       0      (0) STRUCTURE       48  DSPAPQBO
       0      (0) CHARACTER        8   APQBO_SSID      Subsystem identifier
       8      (8) UNSIGNED         4   APQBO_FIRSTUOR  Offset of first UOR entry
      12      (C) UNSIGNED         4   APQBO_LASTUOR   Offset of last UOR entry
      16     (10) CHARACTER       12   APQBO_TIMEFIRST Earliest UOR time
      28     (1C) CHARACTER       12   APQBO_TIMELAST  Latest UOR time
      40     (28) BIT(8)           1   APQBO_FLAGS     Backout flags
                  1... ....             APQBO_SAVER    SAVUOR call during restart
      41     (29) CHARACTER        3   *               Reserved
      44     (2C) SIGNED           4   APQBO_UORCOUNT  Number of UORs

     ================================================================================
     The following structure maps the unit of recovery entries.
     There is one such entry for each unit of recovery (that is, there
     are apqbo_UORcount entries). Each unit of recovery entry
     contains the offset within the backout block to the previous
     and following entries. Field apqbo_PrevUOR is the offset of
     the previous entry and apqbo_NextUOR is the offset of the
     following entry. For the first unit of recovery (UOR) entry,
     apqbo_PrevUOR will be zero. Similarly, apqbo_NextUOR will be
     zero for the last entry. Addressability to the first UOR entry
     is given by:
         rfy apqbo_UORentry based(addr(DSPAPQBO) + apqbo_FirstUOR)
     Addressability to the last UOR entry is given by:
         rfy apqbo_UORentry based(addr(DSPAPQBO) + apqbo_LastUOR)
     Addressability to the next UOR entry, if one exists (that is:
     apqbo_NextUOR not equal to 0), is given by:
         rfy apqbo_UORentry based(addr(DSPAPQBO) + apqbo_NextUOR)
     Similarly, to address the previous entry (when apqbo_PrevUOR
     not equal to 0):
         rfy apqbo_UORentry based(addr(DSPAPQBO) + apqbo_PrevUOR)
     Once addressability has been established to a UOR entry,
     addressability to the ith database for this UOR is given by:
         rfy apqbo_DBent
                     based(addr(apqbo_UORentry) + apqbo_DBoffset
                           + (i-1) apqbo_DBLength)
     ================================================================================
```

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)     DESCRIPTION
======== ======== ========= ========  ============== ==============================
       0      (0) STRUCTURE       64  APQBO_UORENTRY Unit of Recovery entry
       0      (0) CHARACTER       64  APQBO_PREFIX   Prefix section
       0      (0) UNSIGNED         4  APQBO_NEXTUOR  Offset of next UOR entry
       4      (4) UNSIGNED         4  APQBO_PREVUOR  Offset of previous UOR entry
       8      (8) UNSIGNED         4  APQBO_DBOFFSET Offset to DB entries
      12      (C) CHARACTER       12  APQBO_UORTIME  Time stamp for this UOR
      24     (18) CHARACTER        8  APQBO_UORPSB   PSB name
      32     (20) BIT(16)          2  APQBO_UORFLAGS
                  1... ....           APQBO_DEFBO    Deferred backout - dynamic
                                                      backout failure
                  .1.. ....           APQBO_INFLT    Inflight UOR
                  ..1. ....           APQBO_INDOU    Indoubt UOR
                  ...1 ....           APQBO_BMP      BMP UOR
                  .... 1...           APQBO_BOCAN    BBO identified candidate
                  .... .1..           APQBO_COLDN    Cold start ended for UOR
                  .... ..1.           APQBO_BBOK     Backed out OK by BBO
                  .... ...1           APQBO_CMD      UOR modified by command
      33     (21) 1... ....           APQBO_BATCH    Batch IMS UOR
      34     (22) CHARACTER        6  *              Reserved
      40     (28) CHARACTER       16  APQBO_RTOKN    Recovery token
      40     (28) CHARACTER        8  APQBO_RTSSID   SSID for this token
      48     (30) CHARACTER        8  APQBO_UORID    Unique UOR ID
      56     (38) SIGNED           4  APQBO_DBCOUNT  Number of DBs for this UOR
      60     (3C) UNSIGNED         2  APQBO_DBLENGTH Length of each DB entry
      62     (3E) CHARACTER        2  *              Reserved

 OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)     DESCRIPTION
======== ======== ========= ========  ============== ================================
       0      (0) STRUCTURE       16  APQBO_DBENT    Database entry
       0      (0) CHARACTER        8  APQBO_DBNAME   Database name
       8      (8) BIT(8)           1  APQBO_DBFLAGS  Flags
                  1... ....           APQBO_DBOUT    UOR backed out for this DB
                  .1.. ....           APQBO_DBDEF    Dyn backout failure this DB
       9      (9) CHARACTER        7  *              Reserved

CONSTANTS

  LEN  TYPE           VALUE          NAME            DESCRIPTION
```

```
=====  =========  =================  =============== ================================
    8  CHARACTER   DSPAPQBO          APQBO_EYECATCHER
```

Related concepts

“Macro forms of the DSPAPI macro” on page 344
There are four different macro forms for the DSPAPI macro: Standard (S), List (L), Modify (M), and Execute (E), with two variations, COMPLETE and NOCHECK. The List, Modify, and Execute forms are usually used in combinations when writing reentrant programs or when the application issues multiple requests.

Related reference

“DBRC query request (QUERY)” on page 367
You can use the DBRC Query request (DSPAPI FUNC=QUERY) along with the TYPE parameter to retrieve various types of information from the RECON data set.

Database query request (TYPE=DB)

You can use the Database Query request (DSPAPI FUNC=QUERY TYPE=DB) to retrieve information from the RECON concerning one or more registered databases.

This information includes the following database types:

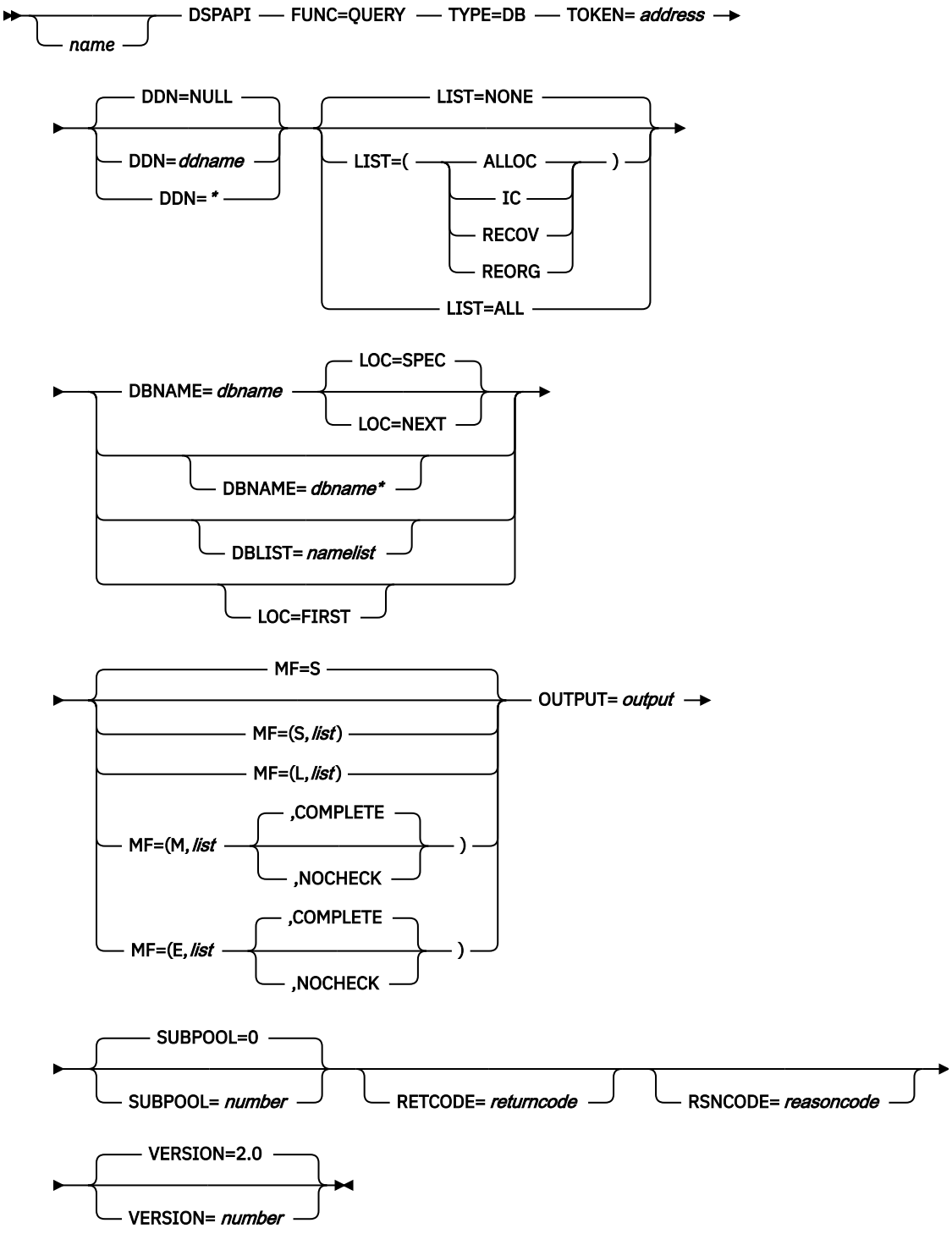
- Full function
- Fast Path DEDB
- HALDB (returns information about the HALDB master and all of its partitions)

Optionally, you can request data set and area information. If you request this information, you can also request recovery-related information for the data set or area, including allocation, image copy, recovery, and reorganization information.

Subsections:

- “Database not found output” on page 393
- “DSECT of DSPAPQNF” on page 394

Syntax for the TYPE=DB query request



Parameters for the TYPE=DB query request

name
If used, begins in column 1.

TYPE=DB
Specifies that RECON information for one or more databases is requested.

TOKEN=*address* | (2 - 12)
Specifies the address of a 4-byte field that was returned on the FUNC=STARTDBRC request.

DBNAME=*symbol* | *symbol | (2 - 12)**
Specifies the name of the database whose information is being queried. This name can be the name of a full-function database, a Fast Path DEDB, or a HALDB. You can use the wildcard keyword * (an asterisk) at the end of a name to query databases whose names match the pattern. In this case, the asterisk must be preceded by at least one alphabetic character.

You must specify a DBNAME or DBLIST if you do not specify LOC=FIRST.

DBLIST=*namelist* | (2 - 12)
Specifies the list of database names whose information is being queried. Each name in the list can be the name of a Full Function database, a Fast Path DEDB, or a HALDB.

The list consists of a header and one or more eight-character list entries. The header consists of a fullword containing the number of entries in the list. The name entries are left-aligned and are padded with blanks.

You must specify a DBNAME or DBLIST if you do not specify LOC=FIRST.

DDN=*ddname* | (2 - 12) | NULL
Specifies the DD name of the data set or area. An asterisk (*) can be specified to return information about all DBDSs or areas for the database. If a specific DD name is specified with DBLIST or a DBNAME that specifies a HALDB, the specific DD name is ignored and treated as if DDN=* was specified.

If you specify DDN=NULL, no DBDS or area information is returned. DDN=NULL is the default.

LIST=(ALLOC | IC | RECOV | REORG) | ALL | LIST=NONE
Specifies the type (or types) of supporting information to be included in the query output for the specified DBDS or area.

If DDN is not specified, this information is returned for all DBDSs or areas in the database.

One or more of the specific values, separated by commas, can be included in the list: ALLOC (allocation records), IC (image copy records), RECOV (recovery records), or REORG (reorganization records). LIST=ALL specifies that all supporting information is requested.

To view information about HALDB online alter processing, specify LIST(REORG). The information about alter processing is included with the information about any other reorganization processes.

If you specify LIST=NONE, no supporting information is returned. LIST=NONE is the default.

LOC=FIRST | NEXT | SPEC
Specifies that the request is for either the specified, the first, or the next database defined in the RECON.

DBNAME=*dbname* is required when you specify LOC=NEXT or LOC=SPEC, but it is not allowed for LOC=FIRST.

Databases are in alphanumeric order. The next database might not necessarily be of the same type as the database name specified in the DBNAME parameter. The value in the DBNAME parameter is used as the base of the search and does not need to be a name of a database registered in the RECON.

LOC=SPEC is the default.

MF=S | L | M | E
Specifies the macro form of the request.

OUTPUT=*output* | (2 - 12)
Specifies a field to receive a pointer to the first block of a possible chain of database information blocks.

The output address is zero if no output was built. This can occur if nothing in the RECON satisfies the request or if an error occurs before any output could be built.

The storage for the output blocks is not pre-allocated by the caller. DBRC acquires storage from the specified subpool for these blocks. The caller must free this storage using the Buffer Release service (DSPAPI FUNC=RELBUF) and specify the returned output address.

SUBPOOL= 0 | *number*
Specifies the subpool number for the storage being obtained. If not specified, the default is the subpool specified by the FUNC=STARTDBRC request. Otherwise, subpool 0 is the default.

RETCODE=*returncode* | (2-12)
If specified as a symbol, specifies the label of a word of storage to receive the return code. If specified as a register, the register must contain the address of a word of storage to receive the return code. Regardless of whether RETCODE is specified, register 15 contains the return code.

RSNCODE=*reasoncode* | (2-12)
If specified as a symbol, the symbol must be the label of a word of storage to receive the reason code. If specified as a register, the register must contain the address of a word of storage to receive the reason code. Regardless of whether RSNCODE is specified, register 0 contains the reason code.

VERSION=2.0 | *number*
Specifies the version number of the parameter list to be generated by this request.

To use the parameters associated with a version, you must specify the number of that version or a later version. If you specify an earlier version level, the parameter is not accepted by macro processing, and an error message is issued at assembly time. If parameters have a version dependency, the parameter descriptions with each request type identify the version number required.

Valid version numbers for the FUNC=QUERY TYPE=DB request are 1.0 and 2.0.

Return and reason codes for the TYPE=DB query request

Table 88. Return and reason codes for TYPE=DB query requests

Code type	Return codes	Reason codes	Meaning
	X'00000000'	X'00000000'	Request completed successfully.
Partial success	X'00000004'	X'D8200001'	One or more databases in DBLIST is not registered in the RECON. A "DB not found" data block with an eyecatcher of DSPAPQNF is built for each database that was not found. The block is in the chain of blocks returned.
Warning	X'00000008'	X'D8220001'	No partitions are registered in RECON data set for the HALDB. No information blocks are returned.
	X'00000008'	X'D8200001'	None of the databases in the DBLIST are registered in the RECON. No information blocks are returned.
	X'00000008'	X'D8220002'	The specified partition is not registered in RECON. No information blocks are returned.
	X'00000008'	X'D8200002'	The specified database is not registered in the RECON. No information blocks are returned. If the request specified LOC=NEXT, you have reached the end of the list of databases registered in the RECON.
	X'00000008'	X'D8200003'	No DBs with the specified wildcard name pattern are registered in RECON. No information blocks are returned.
	X'00000008'	X'D8210002'	The specified DBDS or Area is not registered in the RECON. No information blocks are returned.

Table 88. Return and reason codes for TYPE=DB query requests (continued)

Code type	Return codes	Reason codes	Meaning
	X'00000008'	X'D8220003'	A partition preceding the specified partition does not exist in the RECON data set. No information blocks are returned.
	X'00000008'	X'D8220004'	A partition following the specified partition does not exist in the RECON data set. No information blocks are returned.
Severe error	X'0000000C'	X'C9000001'	Invalid TOKEN. The TOKEN block passed to the API is not recognized as a TOKEN created by a FUNC=STARTDBRC call.
	X'0000000C'	X'C900000A'	The TCB address is not the same as the TCB address under which the STARTDBRC service was issued.
	X'0000000C'	X'D8200100'	Security error. SAF or the DBRC Command Authorization exit routine (DSPDCAX0) determined that the user is not authorized to perform the request.
Storage error	X'00000028'	X'D8200001'	Error obtaining storage for IMSDB block.
	X'00000028'	X'D8200002'	Error obtaining storage for HALDB block.
	X'00000028'	X'D8200003'	Error obtaining storage for FPDEDB block.
	X'00000028'	X'D8200004'	Error obtaining storage for DB not found block (DBNOTFND).
	X'00000028'	X'D8210001'	Error obtaining storage for DBDS block.
	X'00000028'	X'D8210002'	Error obtaining storage for AREA block.
	X'00000028'	X'D8210003'	Error obtaining storage for RCVINFO block.
	X'00000028'	X'D8210004'	Error obtaining storage for ALLOC block.
	X'00000028'	X'D8210005'	Error obtaining storage for IC block.
	X'00000028'	X'D8210006'	Error obtaining storage for REORG block.
	X'00000028'	X'D8210007'	Error obtaining storage for RECOV block.
	X'00000028'	X'D8220001'	Error obtaining storage for PART block.
	X'00000028'	X'D9100001'	An error occurred processing the request. DBRC will release storage that was obtained up to this point. Another error was encountered, however, during the attempt to release storage.
Internal error	X'0000002C'	X'D8000001'	RECON open failure.
	X'0000002C'	X'D8200001'	DB record locate failure processing DBLIST.
	X'0000002C'	X'D8200002'	DB record locate failure processing single database request.
	X'0000002C'	X'D8200003'	DB record locate failure processing database request using a wildcard.
	X'0000002C'	X'D82021xx'	Internal Query DBDS call returned RC=X'30' RSN=X'D82100xx', a parameter error.

Table 88. Return and reason codes for TYPE=DB query requests (continued)

Code type	Return codes	Reason codes	Meaning
	X'0000002C'	X'D82022xx'	Internal Query PART call returned RC=X'30' RSN=X'D82200xx', a parameter error.
	X'0000002C'	X'D8210001'	Failure locating the specified DBDS record.
	X'0000002C'	X'D8210002'	Failure locating the next DBDS record.
	X'0000002C'	X'D8210003'	Failure locating the first DBDS record.
	X'0000002C'	X'D8210004'	Failure locating the first Area AUTH record.
	X'0000002C'	X'D8210005'	Failure locating the first ALLOC record.
	X'0000002C'	X'D8210006'	Failure locating the next ALLOC record.
	X'0000002C'	X'D8210007'	Failure locating the first IC record.
	X'0000002C'	X'D8210008'	Failure locating the next IC record.
	X'0000002C'	X'D8210009'	Failure locating the first REORG record.
	X'0000002C'	X'D821000A'	Failure locating the next REORG record.
	X'0000002C'	X'D821000B'	Failure locating the first RECOV record.
	X'0000002C'	X'D821000C'	Failure locating the next RECOV record.
	X'0000002C'	X'D8220001'	Failure locating the first HALDB partition record.
	X'0000002C'	X'D8220002'	Failure attempting to locate the DB record associated with the HALDB partition being processed.
	X'0000002C'	X'D8220003'	Failure locating the next HALDB partition record.
Parameter error	X'00000030'	X'C9000001'	The function (FUNC) specified in the parameter list passed to the API is invalid.
	X'00000030'	X'C9000002'	Invalid TOKEN field address. The address of the field containing the API TOKEN failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000003'	Invalid RETCODE field address. The address of the field to contain the API RETCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000004'	Invalid RSNCODE field address. The address of the field to contain the API RSNCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000005'	Invalid OUTPUT field address. The address of the field to contain the OUTPUT address failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000008'	Invalid DBNAME or DBLIST address. The address of the field containing the DBNAME or DBLIST failed validity checking. The address specifies storage not owned by the calling program.

Table 88. Return and reason codes for TYPE=DB query requests (continued)

Code type	Return codes	Reason codes	Meaning
	X'00000030'	X'C9000009'	Invalid DDN address. The address of the field containing the DDN failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'D8000001'	Missing or invalid OUTPUT parameter.
	X'00000030'	X'D8200001'	LOC parameter is not allowed with DBLIST.
	X'00000030'	X'D8220001'	The DBNAME or PARTNAME is required.
	X'00000030'	X'D8000002'	Invalid value specified for TYPE parameter.
	X'00000030'	X'D8200002'	DBNAME parameter is not allowed with LOC=FIRST.
	X'00000030'	X'D8220002'	The LOC=FIRST | LAST is required with the DBNAME.
	X'00000030'	X'D8200003'	DBNAME parameter is required with LOC=NEXT.
	X'00000030'	X'D8220003'	The LOC=FIRST | LAST is not allowed with the PARTNAME.
	X'00000030'	X'D8200004'	DBNAME or DBLIST is required.
	X'00000030'	X'D8200005'	Count of databases in DBLIST is zero.
	X'00000030'	X'D8200006'	Database information is being requested for a HALDB partition. DBNAME or DBLIST contains a partition name.
	X'00000030'	X'D8200007'	DBNAME wildcard not allowed with LOC=NEXT
	X'00000030'	X'D8200100'	When using a wildcard, at least one alphabetic character must precede the asterisk.
	X'00000030'	X'D8200101'	When using a wildcard, the asterisk must be the last character.

Output for TYPE=DB query request

The following figures illustrate the format of output from QUERY TYPE=DB requests. The sample DSECTs that follow the figures describe in detail the fields of the storage blocks and their relationship.

Full function output

The following shows the fields of the DSPAPQDB block.

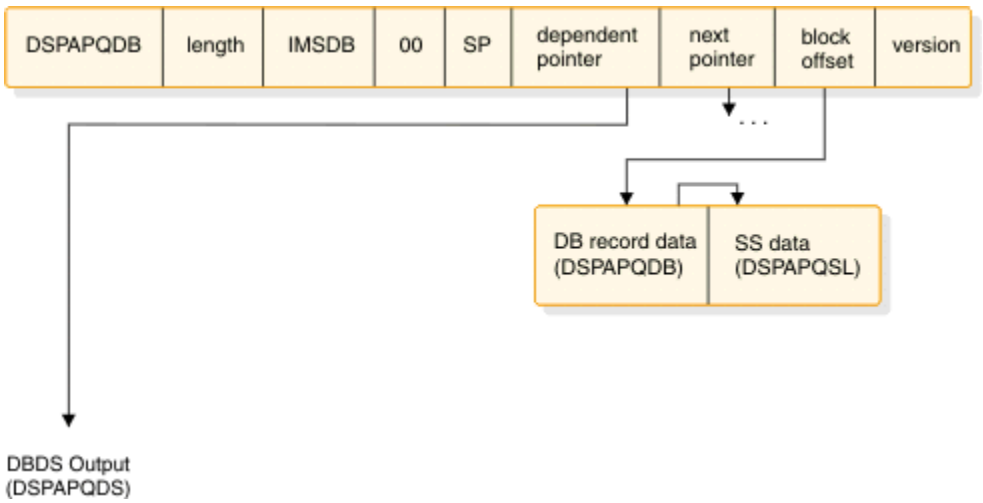


Figure 14. Format of QUERY TYPE=DB (full function) output

The DBDS information is returned only if DDN is specified.

DSECT of DSPAPQDB

The following example describes the fields contained in the DSPAPQDB block shown in Figure 14 on page 380.

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE         LENGTH  NAME (DIM)       DESCRIPTION
======== ======== ========= ========  ==============   ================================
       0      (0) STRUCTURE        96  DSPAPQDB
       0      (0) CHARACTER         8   APQDB_DBNAME    Database name
       8      (8) UNSIGNED          4   APQDB_SSLIST    Offset to SS list (DSPAPQSL), zero
                                                        is no SS auth'd
      12      (C) SIGNED            4   *(3)            Reserved
      24     (18) UNSIGNED          2   APQDB_IRCNT     IC receive needed counter
      26     (1A) BIT(8)            1   APQDB_AUFLAG    Authorization flags
                  1... ....             APQDB_BKFLG     Backout needed flag
                  .1.. ....             APQDB_PAFLG     Prohibit authorization
                  ..1. ....             APQDB_RDFLG     Read only SS auth
                  ...1 ....             APQDB_NONRV     nonrecoverable
                  .... 1...             APQDB_DBREORGI  Reorg intent
                  .... .1..             APQDB_DBQUI     Quiesce in progress
                  .... ..1.             APQDB_DBQUIH    Quiesce held
      27     (1B) CHARACTER         5   APQDB_IRLMAU    IRLMID of auth SS
      32     (20) SIGNED            2   APQDB_RCVCTR    Recovery needed count
      34     (22) SIGNED            2   APQDB_ICCTR     IC needed count
      36     (24) SIGNED            2   APQDB_ICRECCTR  IC recommended counter
      38     (26) UNSIGNED          1   APQDB_SHRLVL    Share level of DB
      39     (27) UNSIGNED          1   APQDB_HELDAU    Held auth state
                  1... ....             APQDB_HAUBIT    High order bit flag
      40     (28) UNSIGNED          2   APQDB_DMBNUM    Global DMB number
      42     (2A) SIGNED            2   APQDB_SSNUM     # of SS auth DB
      44     (2C) UNSIGNED          2   APQDB_SSENTLEN  Length of each SS entry
      46     (2E) UNSIGNED          1   APQDB_CACCSS    Access state for chg auth
      47     (2F) UNSIGNED          1   APQDB_CANCDD    Encode state for chg auth
      48     (30) UNSIGNED          1   APQDB_CAHELD    Held state for chg auth
      49     (31) CHARACTER         3   *               Reserved
      52     (34) UNSIGNED          2   APQDB_EQECNT    Total EQE count
      54     (36) BIT(16)           2   APQDB_RSRFLG    Flags
                  1... ....             APQDB_RCVTRK    Recovery level tracking
                  .1.. ....             APQDB_TRKSPN    Tracking is suspended
                  ..1. ....             APQDB_PURBIT    Suspended by time
                  ...1 ....             APQDB_ICNDIS    IC needed disabled option
      56     (38) CHARACTER         8   APQDB_GSGNAME   Global Service Group name
      64     (40) UNSIGNED          4   APQDB_USID      Last alloc USID
      68     (44) UNSIGNED          4   APQDB_AUSID     Last authorized USID
      72     (48) UNSIGNED          4   APQDB_RUSID     Last received USID
      76     (4C) UNSIGNED          4   APQDB_HUSID     Hardened by tracker USID
      80     (50) UNSIGNED          4   APQDB_RNUSID    Receive needed USID
      84     (54) CHARACTER         8   APQDB_RECOVGRP  Recovery Group name
      92     (5C) CHARACTER         4   *               Reserved

CONSTANTS
```

```
  LEN  TYPE           VALUE          NAME              DESCRIPTION
=====  =========  =================  ===============   ==============================
    8  CHARACTER   DSPAPQDB          APQDB_EYECATCHER
     Possible Quiesce request type values (APQDB_DBQTYPE)
```

DSECT of DSPAPQSL

The following example describes the fields contained in the SS data (DSPAPQSL).

```
DSPAPQSL

  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)      DESCRIPTION
======== ======== ========= ========  ==============  ==============================
       0      (0) STRUCTURE       16  DSPAPQSL
       0      (0) CHARACTER       16  APQSL_SSINFO    Subsystem list entry
       0      (0) CHARACTER        8  APQSL_SSNAME    Subsystem ID
       8      (8) UNSIGNED         1  APQSL_ACCESS    Access intent
       9      (9) UNSIGNED         1  APQSL_NCDDST    Encoded state
      10      (A) BIT(8)           1  APQSL_SSFLGS    Flags
                  1... ....           APQSL_SSROLE    0 - Active SS, 1 - Tracking SS
                  .1.. ....           APQSL_SSXRFC    1 - XRF Capable
                  ..1. ....           APQSL_SSBAT     1 - Batch SS
                  ...1 ....           APQSL_SSIC      1 - IC SS
      11      (B) BIT(8)           1  *               Reserved
      12      (C) CHARACTER        4  APQSL_BKINFO    Backout information
      12      (C) SIGNED           2  APQSL_BKCTR     Backout needed count
      14      (E) SIGNED           2  APQSL_BKNUM     Backout done count

CONSTANTS

  LEN  TYPE           VALUE          NAME              DESCRIPTION
=====  =========  =================  ===============   ==============================

     Possible access intent values (apqsl_ACCESS)
================================================================================
    1  HEX          01                APQSL_ACCRO      READ-GO
    1  HEX          02                APQSL_ACCRD      READ
    1  HEX          03                APQSL_ACCUP      UPDATE
    1  HEX          04                APQSL_ACCEX      EXCLUSIVE
```

Fast Path DEDB output

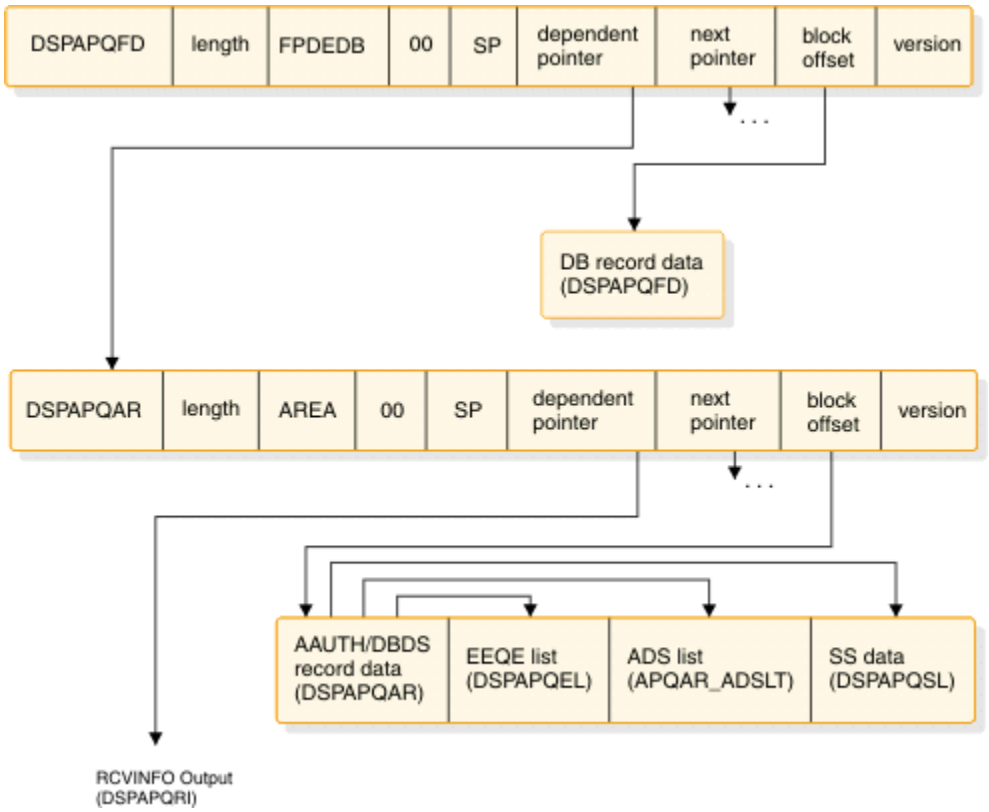


Figure 15. Format of QUERY TYPE=DB (Fast Path DEDB) output

The area information is returned only if DDN is specified. Recovery information (RCVINFO) is only returned if the LIST parameter is specified.

DSECT of DSPAPQFD

The following example describes the fields contained in the DSPAPQFD and DSPAPQAR blocks shown in Figure 15 on page 382.

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)              DESCRIPTION
======== ======== ========= ========  ==============          ============================
       0      (0) STRUCTURE       38  DSPAPQFD
       0      (0) CHARACTER        8  APQFD_DBNAME            Database name
       8      (8) SIGNED           4  *(4)                    Reserved
      24     (18) SIGNED           2  APQFD_RCVCTR            Recovery Needed Counter
      26     (1A) SIGNED           2  APQFD_ICCTR             IC Needed Counter
      28     (1C) SIGNED           2  APQFD_ICRECCTR          IC Recommended Counter
      30     (1E) UNSIGNED         2  APQFD_DMBNUM            Global DMB number
      32     (20) UNSIGNED         2  APQFD_EQECNT            Total EEQE count
      34     (22) SIGNED           2  APQFD_AUTHDAREAS        Number of Areas authorized
      36     (24) UNSIGNED         1  APQFD_SHRLVL            Share Level
      37     (25) BIT(8)           1  APQFD_FLAGS             Flags
                  1... ....           APQFD_PAFLG             Prohibit authorization
                  .1.. ....           APQFD_NONRV             nonrecoverable
                  ..1. ....           APQFD_ICNDIS            IC needed disabled option
                  ...1 ....           APQFD_USRRV             user-recoverable (VERSION=1.01)
                  .... 1...           APQFD_FULLSEG_DEFAULT   Default full segment logging
                                                              setting for areas (VERSION=4.0)
      38     (26) UNSIGNED         2  APQFD_ALTER#            FP DEDB alter status
                                                                 (VERSION=5.0)
      40     (28) CHARACTER        8  APQFD_RANDOMIZER        Randomizer name
                                                                 (VERSION=5.01)
CONSTANTS

  LEN  TYPE           VALUE           NAME              DESCRIPTION
=====  =========  ================  ==============   ============================
    8  CHARACTER   DSPAPQFD          APQFD_EYECATCHER
```

DSECT of DSPAPQAR

The following example describes the fields contained in the DSPAPQFD and DSPAPQAR blocks shown in Figure 15 on page 382.

```
 OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH   NAME (DIM)     DESCRIPTION
======== ======== ========= ========   ============  ============================
       0      (0) STRUCTURE      196   DSPAPQAR
       0      (0) CHARACTER        8   APQAR_DBNAME   Database name
       8      (8) CHARACTER        8   APQAR_AREANAME Area name
      16     (10) UNSIGNED         4   APQAR_EEQELIST Offset to EEQE list
                                                      (DSPAPQEL), zero if no EEQEs
      20     (14) UNSIGNED         4   APQAR_SSLIST   Offset to SS list (DSPAPQSL),
                                                      zero if no SS auth'd
      24     (18) UNSIGNED         4   APQAR_ADSLIST  Offset to ADS list, zero if
                                                      none registered
      28     (1C) SIGNED           4   *              Reserved
      32     (20) UNSIGNED         1   APQAR_SHRLVL   Share level of DB
      33     (21) UNSIGNED         1   APQAR_HELDAU   Held auth state
                  1... ....            APQAR_HAUBIT   High order bit flag
      34     (22) UNSIGNED         2   APQAR_DMBNUM   Global DMB number
      36     (24) SIGNED           2   APQAR_SSNUM    # subsytems authd to Area
      38     (26) UNSIGNED         2   APQAR_SSENTLEN Length of each SS entry
      40     (28) UNSIGNED         1   APQAR_CACCSS   Access state for CHG AUTH
      41     (29) UNSIGNED         1   APQAR_CANCDD   Encoded state for CHG AUTH
      42     (2A) UNSIGNED         1   APQAR_CAHELD   Held state for CHG AUTH
      43     (2B) CHARACTER        5   APQAR_IRLMAU   IRLMID of auth SS
      48     (30) BIT(16)          2   APQAR_FLAGS
                  1... ....            APQAR_RECYC    REUSE image copies
                  .1.. ....            APQAR_ICREC    Image Copy Recommended
                  ..1. ....            APQAR_IC       Image Copy Needed
                  ...1 ....            APQAR_ICNDIS   IC needed disabled option 1 =
                                                      IC Needed Disabled
                  .... 1...            APQAR_RECOV    Recovery needed
                  .... .1..            APQAR_INPRO    HSSP CIC in progress
                  .... ..1.            APQAR_GT240    M/C FP GT240 area DEDB
                  .... ...1            APQAR_VSO      VSO flag
      49     (31) 1... ....            APQAR_PREOP    PREOPEN flag
                  .1.. ....            APQAR_PRELD    PRELOAD flag
                  ..1. ....            APQAR_LKASD    VSO CF buffer lookaside
                  ...1 ....            APQAR_MAS      VSO area resides in multi-area
                                                      CF structure
                  .... 1...            APQAR_RRGAL    REORG since last ALLOC, only
                                                      set if RSR-covered
                  .... .1..            APQAR_TSRAL    TS recov since last ALLOC, only
                                                      set if RSR-covered
                  .... ..1.            APQAR_FULLSEG  Full segment logging in effect
                                                      (VERSION=4.0)
      50     (32) BIT(8)           1   APQAR_RSRFLAGS Remote Site Recovery flags
                  1... ....            APQAR_RCVTRK   Recovery Level Tracking
                  .1.. ....            APQAR_TRKSPN   Tracking was suspended
                  ..1. ....            APQAR_PURBIT   Suspended by time
                  ...1 ....            APQAR_RCVRQ    Receive Required
      51     (33) BIT(8)           1   APQAR_AUFLAG   Authorization flags
                  1... ....            APQAR_PAFLG    Prohibit authorization
                  .1.. ....            APQAR_NONRV    nonrecoverable
                  ..1. ....            APQAR_USSRV    User-recoverable (VERSION=1.01)
                  ...1 ....            APQAR_DBQUI    Quiesce in progress
                  .... 1...            APQAR_DBQUIH   Quiesce held
                  .... .1..            APQAR_DBQUICMD HALDB/DEDB on command
                  .... ..1.            APQAR_SHADOW   Shadow Area (VERSION=6.0)
      52     (34) BIT(8)           1   APQAR_DSORG    Data set organization
                  1... ....            APQAR_VSAM     1 = VSAM, 0 = NON-VSAM
                  .1.. ....            APQAR_INDEX    0 = Non-indexed (OSAM or ESDS),
                                                      1 = Indexed(ISAM or KSDS)
                  ..11 1111             *             Reserved - zeroes
```

The following example describes the fields contained in the DSPAPQFD and DSPAPQAR blocks shown in Figure 15 on page 382.

```
      53     (35) CHARACTER        1   APQAR_DBORG      IMS DB organization
      54     (36) CHARACTER        8   APQAR_GSGNAME    GSG Name
      62     (3E) CHARACTER        2   *                 Reserved
      64     (40) UNSIGNED         4   APQAR_USID       Last ALLOC USID
      68     (44) UNSIGNED         4   APQAR_AUSID      Last authorized USID
      72     (48) UNSIGNED         4   APQAR_RUSID      Last received USID
      76     (4C) UNSIGNED         4   APQAR_HUSID      Hardened USID
      80     (50) UNSIGNED         4   APQAR_RNUSID     Receive needed USID
```

```
      84     (54) CHARACTER        8   APQAR_RECOVGRP   Recovery Group name
      92     (5C) CHARACTER        8   APQAR_CAGRPNAME  Change Accum group name
     100     (64) UNSIGNED         2   APQAR_GENMX      Max number of ICs that
                                                        may be predefined for this area
     102     (66) UNSIGNED         2   APQAR_GENNO      Number of available ICs for this area
     104     (68) UNSIGNED         2   APQAR_USDIC      Number of ICs used
     106     (6A) SIGNED           2   APQAR_EEQECOUNT  EEQE count
     108     (6C) UNSIGNED         2   APQAR_EEQELENGTH EEQE entry length
     110     (6E) UNSIGNED         1   APQAR_NOADS      # of ADS in the area
     111     (6F) UNSIGNED         1   APQAR_AVADS      # of available ADS
     112     (70) UNSIGNED         2   APQAR_ADSLENGTH  ADS entry length
     114     (72) CHARACTER        2   *                Reserved
     116     (74) CHARACTER       40   APQAR_JCL        GENJCL members
     116     (74) CHARACTER        8   APQAR_ICJCL      Image copy member
     124     (7C) CHARACTER        8   APQAR_OIJCL      Online IC member
     132     (84) CHARACTER        8   APQAR_RCJCL      Recovery member
     140     (8C) CHARACTER        8   APQAR_DFJCL      DEFLTJCL member
     148     (94) CHARACTER        8   APQAR_RVJCL      Receive JCL member
     156     (9C) UNSIGNED         2   APQAR_RTPRD      IC retention period
     158     (9E) UNSIGNED         2   APQAR_DSID       IMS data set ID
     160     (A0) UNSIGNED         4   APQAR_DSSN       Data set sequence number
     164     (A4) CHARACTER       16   APQAR_CFST1      VSO CF Structure 1
     180     (B4) CHARACTER       16   APQAR_CFST2      VSO CF Structure 2

  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)     DESCRIPTION
======== ======== ========= ========  =============  ============================
       0      (0) STRUCTURE       56  APQAR_ADSLT    Area Data Set List
       0      (0) CHARACTER        8  APQAR_ADSDD    DDNAME of the ADS
       8      (8) CHARACTER       44  APQAR_ADSDN    DSN of the ADS
      52     (34) BIT(8)           1  APQAR_ADSBT
                  1... ....          APQAR_ADSAV    Avail status of ADS
                  .1.. ....          APQAR_ADSFM    Format status of create util
                  ..1. ....          APQAR_ADSCP    Copy status of create util
      53     (35) CHARACTER        3   *             Reserved

CONSTANTS

  LEN  TYPE           VALUE          NAME             DESCRIPTION
=====  =========  ================  ===============  ============================
    8  CHARACTER   DSPAPQAR          APQAR_EYECATCHER
     Possible Quiesce request type values (APQAR_DBQTYPE)
```

DSECT of DSPAPQEL

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)       DESCRIPTION
======== ======== ========= ========  ==============   ================================
       0      (0) STRUCTURE       13  DSPAPQEL
       0      (0) CHARACTER       13  APQEL_EEQEENTRY  EEQE descriptor entry
       0      (0) BIT(8)           1  APQEL_EQEFG      EEQE flags
                  1... ....          APQEL_ERTL       Toleration error
                  .1.. ....          APQEL_ERRD       Read error
                  ..1. ....          APQEL_ERWT       Write error
                  ...1 ....          APQEL_ERUS       DBRC user modified
                  .... 1...          APQEL_ERPM       DBRC permanent error
                  .... .1..          APQEL_INDT       Indoubt EEQE
                  .... ..1.          APQEL_CIIND      Index CI indicator
       1      (1) CHARACTER        4  APQEL_EQE        EEQE
       5      (5) CHARACTER        8  APQEL_SSID       SSID which owns the EEQE
```

HALDB (master and all partitions) output

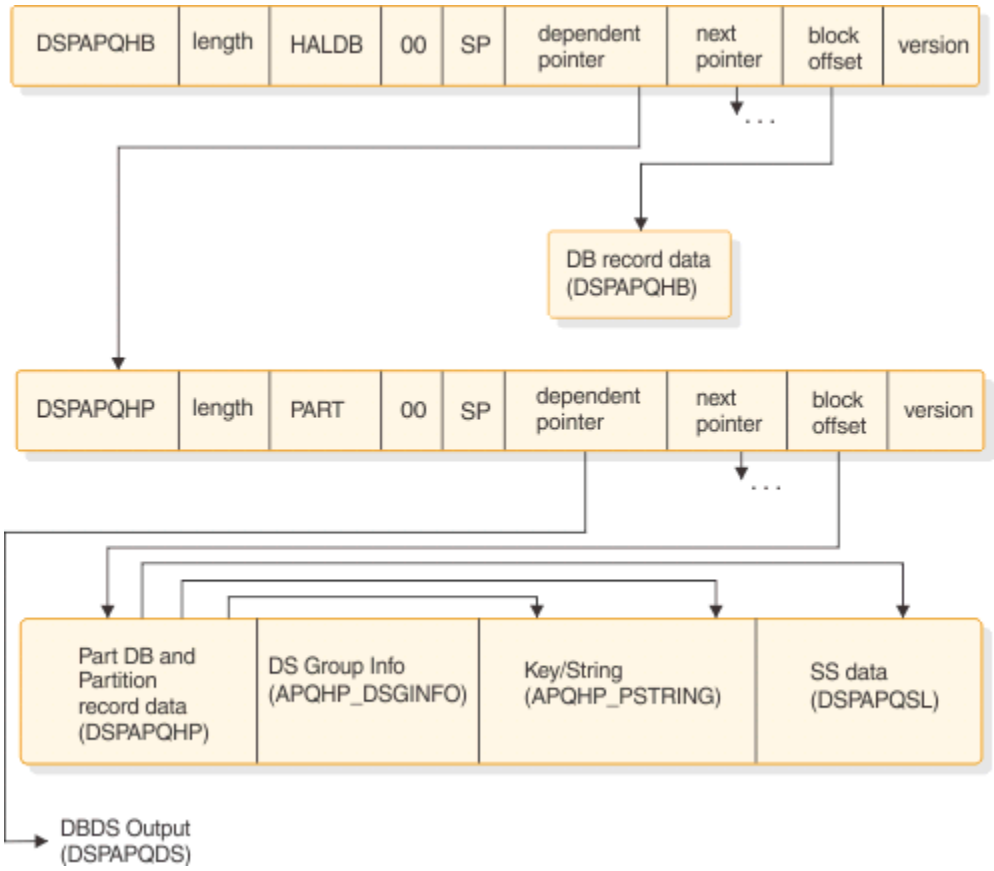


Figure 16. Format of QUERY TYPE=DB (HALDB master and partitions) output

The DBDS information is returned only if DDN is specified.

DSECT of DSPAPQHB

The following two examples describe the fields contained in the DSPAPQHB and DSPAPQHP blocks shown in Figure 16 on page 385. Refer to Figure 17 on page 388 for an illustration of the fields of the DBDS output.

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)      DESCRIPTION
======== ======== ========= ========  ==============  ==============================
       0      (0) STRUCTURE       60  DSPAPQHB
       0      (0) CHARACTER        8   APQHB_DBNAME   HALDB name
       8      (8) SIGNED           4   *(4)           Reserved
      24     (18) BIT(8)           1   APQHB_FLAGS    Flags
                  1... ....            APQHB_NONRV    nonrecoverable
                  .1.. ....            APQHB_ICNDIS   IC needed disabled
                  ..1. ....            APQHB_OLRCAP   HALDB is OLR capable
                 ...1 ....          APQHB_OSAM8G   HALDB OSAM is 8GB addressability
      25     (19) BIT(8)           1   APQHB_ORG      DB organization
                  1... ....            APQHB_PSINDEX  PSINDEX DB
                  .1.. ....            APQHB_PHIDAM   PHIDAM DB
                  ..1. ....            APQHB_PHDAM    PHDAM DB
                  ...1 ....            APQHB_OSAM     OSAM DB
                  .... 1111            *              Reserved
      26     (1A) UNSIGNED         1   APQHB_SHRLVL   Share level
      27     (1B) UNSIGNED         1   APQHB_DSGCNT   # DS Group members
      28     (1C) UNSIGNED         2   APQHB_DMBNUM   Global DMB number
      30     (1E) UNSIGNED         2   APQHB_PARTID   Current Partition ID
      32     (20) SIGNED           2   APQHB_PART#    Number of parts in HALDB
      34     (22) UNSIGNED         2   APQHB_VERSION# Version number
      36     (24) CHARACTER        8   APQHB_PSNAME   Name of Part Sel Routine
      44     (2C) CHARACTER        8   APQHB_GSGNAME  GSG name
      52     (34) CHARACTER        8   APQHB_RECOVGRP Recovery Group name
      60     (3C)  UNSIGNED         2  APQHB_ALTER#   The total number of partitions
```

```
                                                    to be altered in an online
                                                    HALDB database
      62     (3E)  UNSIGNED         2  APQHB_ALTCMP#  The number of partitions that
                                                    an active alter operation has
                                                    completed at the time the
                                                    query call was processed

CONSTANTS

  LEN  TYPE           VALUE           NAME              DESCRIPTION
=====  =========  ================  ===============   ================================
    8  CHARACTER   DSPAPQHB          APQHB_EYECATCHER
```

DSECT of DSPAPQHP

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)          DESCRIPTION
======== ======== ========= ======== ==============      ================================
       0      (0) STRUCTURE      192  DSPAPQHP
       0      (0) CHARACTER        8  APQHP_HALDBNAME     HALDB name
       8      (8) CHARACTER        8  APQHP_PARTNAME      Partition name
      16     (10) UNSIGNED         4  APQHP_SSLIST        Offset to SS list (DSPAPQSL),
                                                          zero if no SS auth'd
      20     (14) UNSIGNED         4  APQHP_KEYSTRING     Offset to KEYSTRING
                                                          (apqhp_PString), zero is no
                                                          key/string
      24     (18) UNSIGNED         4  APQHP_DSGINFOOFFSET Offset to data set group
                                                          information

      28     (1C) SIGNED           4  APQHP_ALTERINFOOFFSET  Offset of HALDB Alter
                                                          information structure,
                                                          APQHP_ALTERINFO, if present.
                                                          Value is 0 if not present.

      32     (20) CHARACTER       44  *
      32     (20) CHARACTER       37  APQHP_DSNBASE       Base Partition DSN
      76     (4C) CHARACTER       18  APQHP_HDAM          PHDAM fields
      76     (4C) CHARACTER        8  APQHP_RMNAME        Randomizing module name
      84     (54) SIGNED           4  APQHP_RBN           Max relative block number
      88     (58) SIGNED           4  APQHP_BYTES         Max # of bytes
      92     (5C) UNSIGNED         2  APQHP_ANCHR         # of root anchor points
      94     (5E) UNSIGNED         1  APQHP_FBFF          Free block frequency factor
      95     (5F) UNSIGNED         1  APQHP_FSPF          Free space percentage factor
      96     (60) UNSIGNED         2  APQHP_PARTID        Partition ID
      98     (62) SIGNED           2  APQHP_PSTLN         Length of Part Key/String,
                                                          apqhp_PString
     100     (64) UNSIGNED         2  APQHP_DSGINFOLEN
                                                          Length of each
                                                          aphp_DSGinfo entry
     102     (66) UNSIGNED         1  APQHP_DSGCNT        DSG count
     103     (67) BIT(8)           1  APQHP_FLAGS         Flags
                  1... ....           APQHP_PINIT         Partition must be initialized
                  .1.. ....           APQHP_ORDBDS        0=A-J/1=M-V DBDS active
                  ..1. ....           APQHP_OLRON         OLR active
                  ...1 ....           APQHP_DISAB         Partition Disabled
                  .... 1...           APQHP_MVDBDS        1 = M-V DBDS exist
                  .... .1..           APQHP_OLRCAP        Partition is OLR capable
                  .... ..1.           APQHP_OLRREL        1 = RELEASE OLR OWNER
                 .... ...1          APQHP_OSAM8G        8GB OSAM addressability
        ================================================================================
         If the Partitioned DB uses high keys, that is, no Partition
         Selection routine, the next two fields are used to sort the
         partitions in key sequence.
        ================================================================================
     104     (68) CHARACTER        8   APQHP_PREV          DDN of previous partition
     112     (70) CHARACTER        8   APQHP_NEXT          DDN of next partition
     120     (78) CHARACTER        8   APQHP_OLRIMS        Owning IMS for OLR
     128     (80) UNSIGNED         2   APQHP_IRCNT         IC receive needed counter
     130     (82) BIT(8)           1   APQHP_AUFLAG        Authorization flags
                  1... ....           APQHP_BKFLG         Backout needed
                  .1.. ....           APQHP_PAFLG         Prohibit authorization
                  ..1. ....           APQHP_RDFLG         Read only SS auth
                  ...1 ....           APQHP_NONRV         nonrecoverable
                  .... 1...           APQHP_DBREORGI      Reorg intent
                  .... .1..           APQHP_DBQUI         Quiesce in progress
                  .... ..1.           APQHP_DBQUIH        Quiesce held
                  .... ...1           APQAR_DBQUICMD      HALDB/DEDB on command
     131     (83) CHARACTER        5   APQHP_IRLMAU        IRLM ID of auth SS
```

```
     136     (88) SIGNED           2   APQHP_RCVCTR       Recovery needed count
     138     (8A) SIGNED           2   APQHP_ICCTR        IC needed count
     140     (8C) UNSIGNED         1   APQHP_SHRLVL       Share level of DB
     141     (8D) UNSIGNED         1   APQHP_HELDAU       Held auth state
                  1... ....            APQHP_HAUBIT       High order bit flag
     142     (8E) UNSIGNED         2   APQHP_DMBNUM       Global DMB number
     144     (90) SIGNED           2   APQHP_SSNUM        # of SS auth DB
     146     (92) UNSIGNED         2   APQHP_SSENTLEN     Length of each SS entry
     148     (94) UNSIGNED         1   APQHP_CACCSS       Access state for chg auth
     149     (95) UNSIGNED         1   APQHP_CANCDD       Encode state for chg auth
     150     (96) UNSIGNED         1   APQHP_CAHELD       Held state for chg auth
     151     (97) CHARACTER        1   *                  Reserved
     152     (98) UNSIGNED         2   APQHP_EQECNT       Total EQE count
     154     (9A) BIT(16)          2   APQHP_RSRFLG       Flags
                  1... ....            APQHP_RCVTRK       Only recov level trackng
                  .1.. ....            APQHP_TRKSPN       Tracking is suspended
                  ..1. ....            APQHP_PURBIT       Suspended by time
                  ...1 ....            APQHP_ICNDIS       IC needed disabled option
                  .... 1...            APQHP_NOHKEY       High key required

                  .... .1..            APQHP_ALTER        Partition being altered
                  .... ..1.            APQHP_ALTCMP       Partition alter completed;
                                                          partition ready for online
                                                          change
     156     (9C) CHARACTER        8   APQHP_GSGNAME      GSG name
     164     (A4) UNSIGNED         4   APQHP_USID         Last alloc USID
     168     (A8) UNSIGNED         4   APQHP_AUSID        Last authorized USID
     172     (AC) UNSIGNED         4   APQHP_RUSID        Last received USID
     176     (B0) UNSIGNED         4   APQHP_HUSID        Hardened by tracker USID
     180     (B4) UNSIGNED         4   APQHP_RNUSID       Receive needed USID
     184     (B8) SIGNED           2   APQHP_ICRECCTR     IC Recommended Counter
     186     (BA) UNSIGNED         2   APQHP_VERSION#     Version number
     188     (BC) UNSIGNED         1   APQHP_OLRACTHARDCTR
                                                          OLR curs active count
     189     (BD) UNSIGNED         1   APQHP_OLRINACTHARDCTR
                                                          OLR curs inact count
     190     (BE) UNSIGNED         2   APQHP_REORG#       Partition reorg #
     192     (C0) CHARACTER        8   APQHP_OLRBytes     OLR Bytes moved
     200     (C8) CHARACTER        8   APQHP_OLRSegs      OLR Segments moved
     208     (D0) CHARACTER        4   APQHP_OLRRoots     OLR Root Segments

     212     (D4) UNSIGNED         2   APQHP_ALTERINFOLEN Length of each
                                                          apqhp_AlterInfo entry

  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)     DESCRIPTION
======== ======== ========= ========  ============== ================================
       0      (0) STRUCTURE         2  APQHP_DSGINFO  Data set group information
       0      (0) UNSIGNED          2  APQHP_BLKSZ    DS block size, OSAM only

  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)     DESCRIPTION
======== ======== ========= ========  ============== ================================
       0      (0) STRUCTURE         *  APQHP_PSTRING  Partition Key/String

  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)     DESCRIPTION
======== ======== ========= ========  ============== ================================
       0      (0) STRUCTURE         2  APQHP_ALTERINFO  DB Alter information
                                                        Structure present only when an
                                                        alter operation is in progress
       0      (0) UNSIGNED          2  APQHP_ALTERSZ  If alter changes block or CI
                                                        sizes, ALTERSZ contains the
                                                        OSAM block size or VSAM CI size
                                                        for the output data sets of an
                                                        alter operation

CONSTANTS

  LEN  TYPE           VALUE           NAME              DESCRIPTION
=====  =========  ================  ================  ================================
    8  CHARACTER   DSPAPQHP          APQHP_EYECATCHER
```

DBDS output

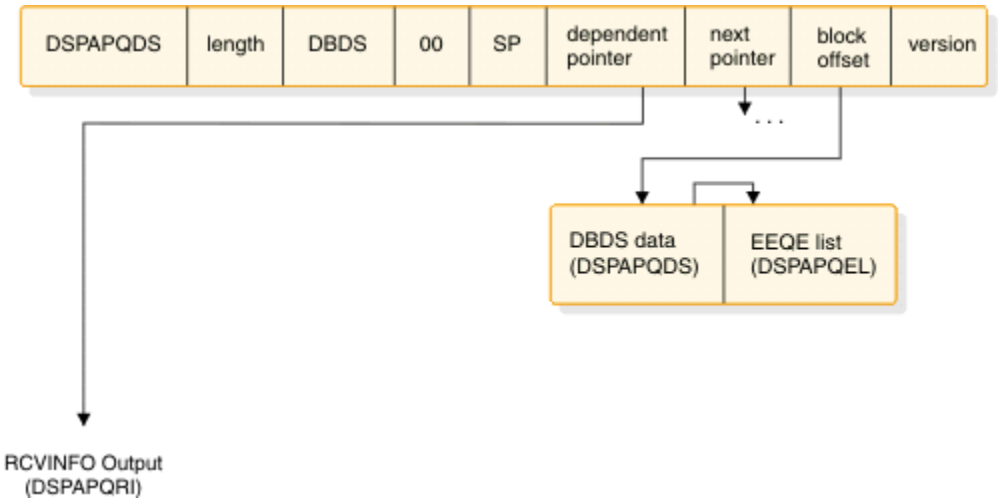


Figure 17. Format of QUERY TYPE=DB (DBDS) output

Recovery information (RCVINFO) is returned only if the LIST parameter is specified.

DSECT of DSPAPQDS

The following example and "DSECT of DSPAPQEL" on page 384 describe the fields contained in the DSPAPQDS block shown in Figure 17 on page 388. Refer to Figure 18 on page 390 for an illustration of the Recovery Information output fields.

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)       DESCRIPTION
======== ======== ========= ======== ==============   ==============================
       0      (0) STRUCTURE      160  DSPAPQDS
       0      (0) CHARACTER        8  APQDS_DBNAME     Database name
       8      (8) CHARACTER        8  APQDS_DDNAME     DD name
      16     (10) UNSIGNED         4  APQDS_EEQELIST   Offset to EEQE list
                                                       (DSPAPQEL), zero if no EEQEs
      20     (14) SIGNED           4  *(3)             Reserved
      32     (20) CHARACTER       44  APQDS_DSN        Data set name
      76     (4C) UNSIGNED         2  APQDS_RTPRD      IC retention period
      78     (4E) UNSIGNED         2  APQDS_DSID       Data set ID number
      80     (50) UNSIGNED         4  APQDS_DSSN       Data set sequence number
      84     (54) UNSIGNED         4  APQDS_RUSID      Recovered-to USID(trkr)
      88     (58) BIT(8)           1  APQDS_FLAGS      BINARY ZEROS
                  1... ....           APQDS_RECYC      REUSE image copies
                  .1.. ....           APQDS_ICREC      Image Copy Recommended
                  ..1. ....           APQDS_RCVRQ      Receive required
                  ...1 ....           APQDS_IC         Image Copy Needed
                  .... 1...           APQDS_RECOV      Recovery Needed
                  .... .1..           APQDS_NONRV      nonrecoverable
      89     (59) BIT(8)           1  APQDS_DSORG      Data set organization
                  1... ....           APQDS_VSAM       1 = VSAM, 0 = NON-VSAM
                  .1.. ....           APQDS_INDEX      0 = Non-indexed (OSAM or ESDS),
                                                       1 = Indexed(ISAM or KSDS)
                  ..11 1111           *                Reserved - zeroes
      90     (5A) CHARACTER        1  APQDS_DBORG      IMS DB organization
      91     (5B) UNSIGNED         1  *                Reserved
      92     (5C) UNSIGNED         2  APQDS_GENMX      Max number of ICs that may be
                                                       predefined for this area
      94     (5E) UNSIGNED         2  APQDS_AVAILIC#
                                                       Number of available ICs for
                                                       this area
      96     (60) UNSIGNED         2  APQDS_USEDIC#    Number of ICs used
      98     (62) SIGNED           2  APQDS_EEQECOUNT
                                                       EEQE count
     100     (64) UNSIGNED         2  APQDS_EEQELENGTH
                                                       EEQE entry length
     102     (66) BIT(8)           1  APQDS_FLG1       Flags
                  1... ....           APQDS_RRGAL      REORG since last ALLOC, only
                                                       set if RSR-covered
                  .1.. ....           APQDS_TSRAL      TS recov since last ALLOC, only
                                                       set if RSR-covered
```

```
     103     (67) BIT(8)           1  APQDS_FLG2       DBDS type flags
                  1... ....           APQDS_PART       TYPEPART record
                  .1.. ....           APQDS_PDATA      TYPEPART subtype DATA
                  ..1. ....           APQDS_PILE       TYPEPART subtype ILE
                  ...1 ....           APQDS_PINDX      TYPEPART subtype Index
     104     (68) CHARACTER        8  APQDS_CAGRPNAME
                                                       Change Accum group name
     112     (70) CHARACTER       40  APQDS_JCL        GENJCL members
     112     (70) CHARACTER        8  APQDS_ICJCL      Image copy member
     120     (78) CHARACTER        8  APQDS_OIJCL      Online IC member
     128     (80) CHARACTER        8  APQDS_RCJCL      Recovery member
     136     (88) CHARACTER        8  APQDS_DFJCL      DEFLTJCL member
     144     (90) CHARACTER        8  APQDS_RVJCL      Receive JCL member
     152     (98) CHARACTER        8  APQDS_ODDN       OLR partner DBDS

CONSTANTS

  LEN  TYPE          VALUE          NAME             DESCRIPTION
=====  =========  ================  ================ ==============================
    8  CHARACTER   DSPAPQDS          APQDS_EYECATCHER
```

Recovery Information (RCVINFO) output

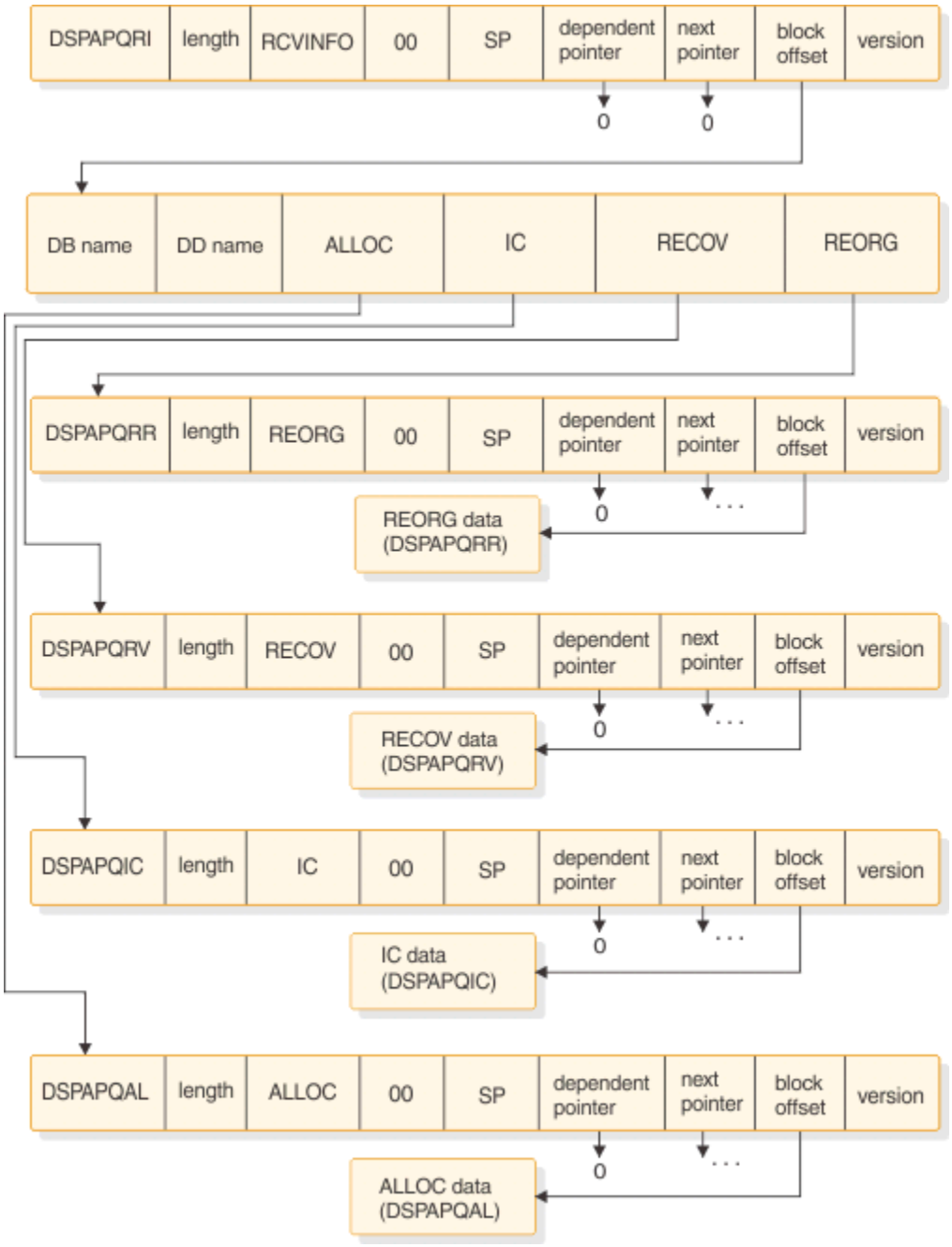


Figure 18. Format of QUERY TYPE=DB (RCVINFO) output

Recovery information (RCVINFO) is returned only if the LIST parameter is specified. The pointers are zero if either the specific information does not exist or it was not requested.

DSECT of DSPAPQRI

The following DSECT example describes the fields that are contained in the DSPAPQRI block as shown in Figure 18 on page 390.

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE         LENGTH  NAME (DIM)       DESCRIPTION
======== ======== ========= ========  ==============   ================================
       0      (0) STRUCTURE         32  DSPAPQRI
       0      (0) CHARACTER          8  APQRI_DBNAME     Database name
       8      (8) CHARACTER          8  *
       8      (8) CHARACTER          8  APQRI_DDNAME     DD name
```

```
       8      (8) CHARACTER        8  APQRI_AREANAME   Area name
      16     (10) ADDRESS          4  APQRI_ALLOCPTR   ptr to ALLOC chain
      20     (14) ADDRESS          4  APQRI_ICPTR      ptr to IC chain
      24     (18) ADDRESS          4  APQRI_RECOVPTR   ptr to RECOV chain
      28     (1C) ADDRESS          4  APQRI_REORGPTR   ptr to REORG chain

CONSTANTS

  LEN  TYPE           VALUE           NAME              DESCRIPTION
=====  =========  ================  ===============   ================================
    8  CHARACTER   DSPAPQRI          APQRI_EYECATCHER
```

DSECT of DSPAPQAL

The following DSECT example describes the fields that are contained in the DSPAPQAL block as shown in Figure 18 on page 390.

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)      DESCRIPTION
======== ======== ========= ========  ==============  ================================
       0      (0) STRUCTURE       88  DSPAPQAL
       0      (0) CHARACTER        8  APQAL_DBNAME    Database name
       8      (8) CHARACTER        8  *
       8      (8) CHARACTER        8  APQAL_DDNAME    DD name or
       8      (8) CHARACTER        8  APQAL_AREANAME  Area name
      16     (10) CHARACTER       12  APQAL_ALLOCTM   Allocation time
      28     (1C) CHARACTER       12  APQAL_DALTM     Deallocation time
      40     (28) CHARACTER       12  APQAL_STRTM     Log start time
      52     (34) UNSIGNED         4  APQAL_DSSN      Field for DSSN value
      56     (38) UNSIGNED         4  APQAL_USID      Update set identifier
      60     (3C) CHARACTER        8  APQAL_ALRID     LRID of begin-upd rec
      68     (44) CHARACTER        8  APQAL_DLRID     LRID of end-upd rec
      76     (4C) CHARACTER        8  APQAL_SLRID     Last LRID applied if suspended
      84     (54) BIT(8)           1  APQAL_FLAGS     Flags
                  1... ....           APQAL_TSUSP     Tracking is suspended
                  .1.. ....           APQAL_NAPPL     No records applied
                  ..1. ....           APQAL_CICPT     Fuzzy ic purge time
                  ...1 ....           APQAL_DBQUI     Quiesce caused deallocation
      85     (55) CHARACTER        3  *               Reserved
CONSTANTS

  LEN  TYPE           VALUE           NAME              DESCRIPTION
=====  =========  ================  ===============   ================================
    8  CHARACTER   DSPAPQAL          APQAL_EYECATCHER
```

DSECT of DSPAPQIC

The following DSECT example describes the fields that are contained in the DSPAPQIC block as shown in Figure 18 on page 390.

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)       DESCRIPTION
======== ======== ========= ========  ==============   ================================
       0      (0) STRUCTURE       64  DSPAPQIC
       0      (0) CHARACTER        8  APQIC_DBNAME     Database name
       8      (8) CHARACTER        8  *
       8      (8) CHARACTER        8  APQIC_DDNAME     DD name or
       8      (8) CHARACTER        8  APQIC_AREANAME   Area name
      16     (10) CHARACTER       12  APQIC_STARTIME   IC start time, packed decimal
      28     (1C) CHARACTER       12  APQIC_STOPTIME   IC stop time, packed decimal
      40     (28) BIT(8)           1  APQIC_TYPE       IMAGE COPY TYPE
                  1... ....           APQIC_BATCH      BATCH
                  .1.. ....           APQIC_CIC        CONCURRENT
                  ..1. ....           APQIC_USERIC     USER IMAGE COPY
                  ...1 ....           APQIC_ONLINE     ONLINE
                  .... 1...           APQIC_SMSIC      SMS IC w/ DB exclusive
                  .... .1..           APQIC_SMSCC      SMS IC w/ DB shared
                  .... ..1.           APQIC_SMSOF      FastRep IC w/ DB exclusive
                                                       (VERSION=2.00)
                  .... ..1            APQIC_SMSON      FastRep IC w/ DB shared
                                                       (VERSION=2.00)
      41     (29) BIT(8)           1  APQIC_STATUS     IC status flags
                  1... ....           APQIC_AVAIL      Available IC
                  .1.. ....           APQIC_IC1        Image Copy 1 exists
                  ..1. ....           APQIC_IC2        Image Copy 2 exists
                  ...1 ....           APQIC_ERR1       Error on image 1
```

```
                .... 1...           APQIC_ERR2       Error on image 2
                .... .1..           APQIC_EMP2       Image 2 defined and unused
     42     (2A) BIT(8)           1  APQIC_FLAGS
                1... ....           APQIC_HSINP      HSSP CIC in progress
                .1.. ....           APQIC_CAT        Catalogued IC (HSSP)
     43     (2B) CHARACTER        1  APQIC_MoreTYPEs  More Image Copy types
                1... ....           APQIC_UserCIC    User Concurrent Image Copy
                                                     (VERSION=2.0)
     44     (2C) CHARACTER        2  *
     44     (2C) UNSIGNED         2  APQIC_OFF1       Offset to image 1 data
     44     (2C) UNSIGNED         2  APQIC_OFFU       Offset to user IC data
     46     (2E) UNSIGNED         2  APQIC_OFF2       Offset to image 2 data
     48     (30) UNSIGNED         4  APQIC_CNT12      Record count
     52     (34) UNSIGNED         4  APQIC_USID       Update set ID
     56     (38) CHARACTER        2  *
     56     (38) UNSIGNED         2  APQIC_LEN12      Length of image 1/2 data
     56     (38) UNSIGNED         2  APQIC_LENU       Length of user IC data
     58     (3A) CHARACTER        6  *                Reserved
     64     (40) UNSIGNED         2  APQIC_OFFUD      Offset to user data (Version=4.0)
     66     (42) UNSIGNED         2  APQIC_LENUD      Length of user data (Version=4.0)
```

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)          DESCRIPTION
======== ======== ========= ======== ==============      ========================
       0      (0) STRUCTURE       64  APQIC_IC12          Data for image 1 or 2
       0      (0) CHARACTER       44  APQIC_DSN12         Data set name
      44     (2C) UNSIGNED         2  APQIC_FILE          File sequence number
      46     (2E) CHARACTER        8  APQIC_RUT12         Unit device type
      54     (36) UNSIGNED         2  APQIC_VOLCT         # of volumes predefined
      56     (38) UNSIGNED         2  APQIC_VOLUS         # of volumes used
      58     (3A) UNSIGNED         2  APQIC_VOLLISTLEN    Length of each volume list
                                                          entry in apqic_VOLS
      60     (3C) UNSIGNED         4  APQIC_VOLLISTOFFSET Offset to volume list

  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)      DESCRIPTION
======== ======== ========= ======== ==============  ==============================
       0      (0) CHARACTER        6   APQIC_VOLS     List of VOLSERs

  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)      DESCRIPTION
======== ======== ========= ======== ==============  ==============================
       0      (0) STRUCTURE       80  APQIC_USER      Data for user IC
       0      (0) CHARACTER       80  APQIC_UDATA     User supplied data

CONSTANTS

  LEN  TYPE           VALUE           NAME              DESCRIPTION
=====  =========  =================  ===============   ==============================
    1  DECIMAL          255          APQIC_MAXV        max # volumes
    8  CHARACTER   DSPAPQIC          APQIC_EYECATCHER
```

DSECT of DSPAPQRV

The following DSECT example describes the fields that are contained in the DSPAPQRV block as shown in Figure 18 on page 390.

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)       DESCRIPTION
======== ======== ========= ======== ==============   ==============================
       0      (0) STRUCTURE       49  DSPAPQRV
       0      (0) CHARACTER        8  APQRV_DBNAME     Database name
       8      (8) CHARACTER        8  *
       8      (8) CHARACTER        8  APQRV_DDNAME     DD name or
       8      (8) CHARACTER        8  APQRV_AREANAME   Area name
      16     (10) CHARACTER       12  APQRV_RUNTIME    The time at which the DBDS was
                                                       recovered
      28     (1C) CHARACTER       12  APQRV_ENDTIME    Partial recovery only, the time
                                                       to which the DBDS was restored
      40     (28) UNSIGNED         4  APQRV_FUSID      First undone USID
      44     (2C) UNSIGNED         4  APQRV_LUSID      Last undone USID
      48     (30) BIT(8)           1  APQRV_FLAGS      Flags
                1... ....           APQRV_PITR       Point In Time Recovery
                .1.. ....        1  APQRV_EXTCM      External command (Version=4.0)
      49     (31) UNSIGNED         1  *                Reserved
      50     (32) UNSIGNED         2  APQRV_OFFUD      Offset to user data (Version=4.0)
      52     (34) UNSIGNED         2  APQRV_LENUD      Length of user data (Version=4.0)
      54     (33) UNSIGNED         2  APQRV_PREORG     Prior reorg number (Version=4.0)
```

```
      56     (36) UNSIGNED         2  APQRV_NREORG     New reorg number (Version=4.0)

   OFFSET   OFFSET
  DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)      DESCRIPTION
 ======== ======== ========= ========  ==============  ================================
        0      (0) STRUCTURE       80  APQRV_USER      Data for user data
        0      (0) CHARACTER       80  APQRV_UDATA     User supplied data (VERSION=4.0)
CONSTANTS

  LEN  TYPE           VALUE           NAME              DESCRIPTION
=====  =========  ================  ===============   ================================
    8  CHARACTER   DSPAPQRV          APQRV_EYECATCHER
```

DSECT of DSPAPQRR

The following DSECT example describes the fields that are contained in the DSPAPQRR block as shown in Figure 18 on page 390.

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)       DESCRIPTION
======== ======== ========= ========  ==============   ================================
       0      (0) STRUCTURE       60  DSPAPQRR
       0      (0) CHARACTER        8  APQRR_DBNAME     Database name
       8      (8) CHARACTER        8  *
       8      (8) CHARACTER        8  APQRR_DDNAME     DD name or
       8      (8) CHARACTER        8  APQRR_AREANAME   Area name
      16     (10) CHARACTER       12  APQRR_RUNTIME    The time at which the DBDS was
                                                       reorganized
      28     (1C) CHARACTER       12  APQRR_STOPTIME   Stoptime of online reorg
      40     (28) BIT(8)           1  APQRR_FLAGS
                  1... ....          APQRR_ONL        1=ONLINE/0=OFFLINE reorg
                  .1.. ....          APQRR_RECOV      1=May be used for recovery

                  ..1. ....          APQRR_ALTER      1=HALDB structure altered by
                                                       an online reorganization
      41     (29) CHARACTER        3  *                Reserved
      44     (2C) UNSIGNED         4  APQRR_USID       Associated USID
      48     (30) CHARACTER       12  APQRR_PITR       Stoptime moved - PITR
      60     (3C) UNSIGNED         4  APQRR_PRAPs      Total number of RAPs processed
      64     (40) UNSIGNED         4  APQRR_Roots      Total number of roots processed
      68     (44) UNSIGNED         2  APQRR_OFFUD      Offset to user data (VERSION=4.0)
      70     (46) UNSIGNED         2  APQRR_LENUD      Length of user data (VERSION=4.0)

   OFFSET   OFFSET
  DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)      DESCRIPTION
 ======== ======== ========= ========  ==============  ================================
        0      (0) STRUCTURE       80  APQRR_USER      Data for user data
        0      (0) CHARACTER       80  APQRR_UDATA     User supplied data (VERSION=4.0)
CONSTANTS

  LEN  TYPE           VALUE           NAME              DESCRIPTION
=====  =========  ================  ===============   ================================
    8  CHARACTER   DSPAPQRR          APQRR_EYECATCHER
```

Database not found output

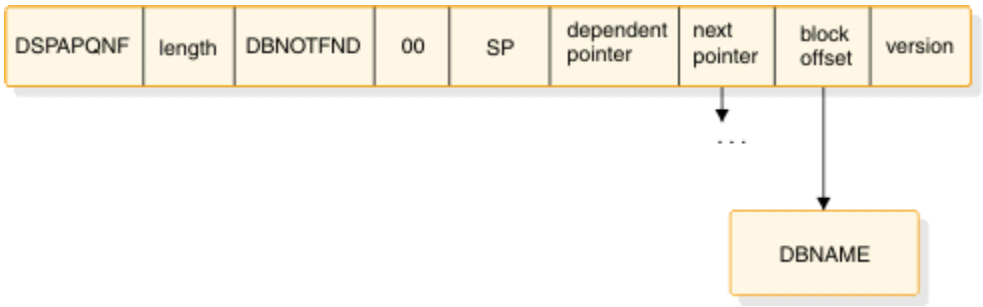


Figure 19. Format of QUERY TYPE=DB (database not found) output

This output block is returned when some of the databases specified in the DBLIST block could not be found in the RECON. One block is returned for each database that could not be found. The database name is included in the data area of this block.

When a database is not found, the macro call receives a return code of four (RC=4). If none of the databases in the list are found (RC=8), no output blocks are returned.

DSECT of DSPAPQNF

The following example describes the fields contained in the DSPAPQNF block shown in Figure 19 on page 393.

```
OFFSET     OFFSET
DECIMAL      HEX   TYPE        LENGTH    NAME (DIM)     DESCRIPTION
======== ========  =========  ========   ============== ==============================
       0      (0)  STRUCTURE          8  DSPAPQNF
       0      (0)  CHARACTER          8  APQNF_DBNAME   DB name
```

Related concepts

"DBRC API" on page 341
Your applications can obtain services from Database Recovery Control (DBRC) through the DBRC application programming interface (API), a release-independent, assembler macro interface. The application obtains these services by issuing DBRC API requests to DBRC, and DBRC returns the results to an area in storage where the application can retrieve them.

Related reference

"DBRC query request (QUERY)" on page 367
You can use the DBRC Query request (DSPAPI FUNC=QUERY) along with the TYPE parameter to retrieve various types of information from the RECON data set.

"DBDS query request (TYPE=DBDS)" on page 394
You can use the DSPAPI FUNC=QUERY TYPE=DBDS request to retrieve information from the RECON data set for one or more DBDSs in a non-HALDB database, a HALDB partition, a DBDS group, or a CA group. You can also request recovery related information for the data set, including allocation, image copy, recovery, and reorganization information.

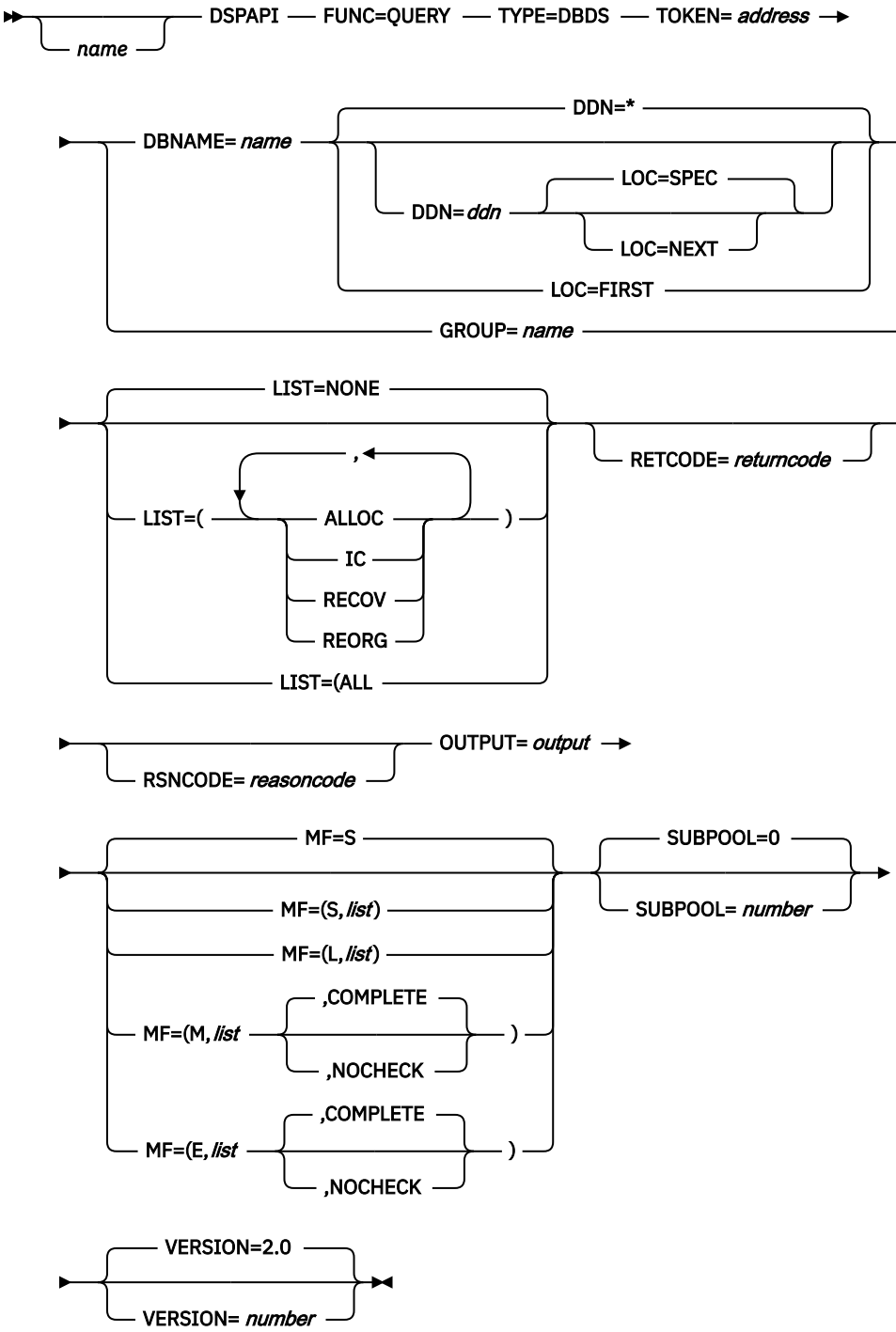
"HALDB partition query request (TYPE=PART)" on page 420
You can use the DSPAPI FUNC=QUERY TYPE=PART request to retrieve information for a particular HALDB partition from the RECON data set. You can request data set information for a specific DBDS or all DBDSs in the partition, and can optionally request recovery-related information for the data set, including allocation, image copy, recovery, and reorganization information.

DBDS query request (TYPE=DBDS)

You can use the DSPAPI FUNC=QUERY TYPE=DBDS request to retrieve information from the RECON data set for one or more DBDSs in a non-HALDB database, a HALDB partition, a DBDS group, or a CA group. You

can also request recovery related information for the data set, including allocation, image copy, recovery, and reorganization information.

Syntax for the TYPE=DBDS query request



Parameters for the TYPE=DBDS query request

name
If used, begins in column 1.

TYPE=DBDS
Specifies that RECON information for a DBDS or area is requested.

Specifying DBDS with the TYPE parameter requires that you also specify a minimum version number of VERSION=2.0.

TOKEN=*address* | (2 - 12)
Specifies the address of a 4-byte field to receive the API token. This token must be included in all subsequent requests associated with this FUNC=STARTDBRC request.

DBNAME=*name* | (2 -12)
Specifies the database name (non-HALDB) or partition name of the DBDS being queried. This parameter is used when you are interested in a specific DBDS of a database or partition. Specifying a HALDB name is not allowed.

DBDNAME OR PARTNAME must be specified.

GROUP=*name* | (2 -12)
Specifies the name of a DBDS group or CA group containing the names of the DBDSs being queried. The LOC or DDN parameters are not allowed with this parameter.

DBNAME or GROUP must be specified. Database and recovery group names are not allowed.

The wildcard GROUP=* can be used for all TYPE=*xxxx*GROUP queries, but not for TYPE=DBDS. However, it can be used for TYPE=DBDSGROUP.

DDN=*ddname* | (2 -12)
Specifies the DD name of the data sets or areas. This parameter is used in conjunction with the DBNAME parameter and the LOC parameter to query a specific data set or the next or previous data set in the database or partition. DDN=* returns information for all of the data sets or areas of the database. DDN=* is the default when DBNAME is specified without LOC=FIRST.

DDN must be specified with LOC=SPEC|NEXT.

When querying the next or previous data set, the value in the DDN parameter is used as the base of the search and does not need to be a DD name of a DBDS registered in RECON for the database or partition.

LIST=NONE | LIST=({ALLOC},{IC},{RECOV},{REORG}) | LIST=ALL
Specifies the types of supporting information to be included in the query output for the returned DBDS. One or more of the specific values is included in the list - ALLOC (allocation records), IC (image copy records), RECOV (recovery records), or REORG (reorganization records). LIST=ALL is specified if all supporting information is requested. LIST=NONE is specified if no supporting information is requested.

LIST=NONE is the default.

LOC=FIRST | NEXT | SPEC
(Optional) - Specifies that the request is for the specified partition (SPEC), the first, or the next DBDS defined in RECON for the database or partition. DBNAME is required with the LOC parameter. DDN, with no wildcard, is required with LOC=NEXT|SPEC. DDN is not allowed with LOC=FIRST. LOC=SPEC is the default when a specific DDN is specified. LOC is not allowed with GROUP.

Specify DDN=* to also request data set or area information. If you specify a specific DDN (DDN=ddn), the specified DDN is ignored and all data set or area information is returned.

Partitions are returned in high key order if the HALDB uses high keys. Otherwise, partitions are returned in alphanumerical order.

RETCODE=*returncode* | (2-12)
If specified as a symbol, specifies the label of a word of storage to receive the return code. If specified as a register, the register must contain the address of a word of storage to receive the return code. Regardless of whether RETCODE is specified, register 15 contains the return code.

RSNCODE=*reasoncode* | (2-12)
If specified as a symbol, the symbol must be the label of a word of storage to receive the reason code. If specified as a register, the register must contain the address of a word of storage to receive the reason code. Regardless of whether RSNCODE is specified, register 0 contains the reason code.

OUTPUT=*output* | (2 - 12)
Specifies a field to receive a pointer to the first of a possible chain of blocks that contain the information for the partition.

The output address is zero if no output was built. This result can occur if nothing in the RECON satisfies the request or if an error occurs before any output could be built.

The storage for the output blocks is not preallocated by the caller. DBRC acquires storage from the specified subpool for these blocks. The caller must free this storage using the Buffer Release service (DSPAPI FUNC=RELBUF) and specify the returned output address.

SUBPOOL= 0 | *number*
Optional parameter that specifies the subpool number for the storage being obtained. If not specified, the default is the subpool specified by the FUNC=STARTDBRC request. Otherwise, subpool 0 is the default.

MF=S | L | M | E
Specifies the macro form of the request.

VERSION=2.0 | *number*
Optional parameter that specifies the version number of the parameter list to be generated by this macro.

To use the parameters associated with a version, you must specify the number of that version or a later version. If you specify an earlier version level, the parameter is not accepted by macro processing, and an error message is issued at assembly time. If parameters have a version dependency, the parameter descriptions with each request type identify the version number required.

The default version is 2.0.

Note: TYPE=DBDS requires that you specify a minimum version number of API VERSION=2.0.

Return and reason codes for the TYPE=DBDS query request

Table 89. DSPAPI FUNC=QUERY TYPE=DBDS query return and reason codes

Code type	Return code	Reason code	Meaning
	X'00000000'	X'00000000'	Request completed successfully.
Warning	X'00000008'	X'D8210001'	No DBDSs or areas for the DB are registered in the RECON data set. No information blocks are returned.
	X'00000008'	X'D8210002'	The specified DBDS or area is not registered in the RECON data set. No information blocks are returned.
	X'00000008'	X'D8210003'	The specified DBNAME is a HALDB. DBNAME must specify a non-HALDB or a HALDB partition name. No information blocks are returned.
	X'00000008'	X'D8210004'	The specified group is not registered in the RECON data set. No information blocks are returned.
	X'00000008'	X'D8210005'	The specified group is not a DBDS or CA group. No information blocks are returned.
	X'00000008'	X'D8210006'	The specified DBNAME is not registered in RECON. No information blocks are returned.
Severe error	X'0000000C'	X'C9000001'	Invalid TOKEN. The TOKEN block passed to the API is not recognized as a TOKEN created by a FUNC=STARTDBRC call.
	X'0000000C'	X'C900000A'	The TCB address is not the same as the TCB address under which the STARTDBRC service was issued.

Table 89. DSPAPI FUNC=QUERY TYPE=DBDS query return and reason codes (continued)

Code type	Return code	Reason code	Meaning
	X'0000000C'	X'D8210001'	SAF or the DBRC cmd auth exit (DSPDCAX0) has determined that the user is not authorized to perform the request.
Storage error	X'00000028'	X'D8210001'	Error obtaining storage for DBDS block.
	X'00000028'	X'D8210002'	Error obtaining storage for AREA block.
	X'00000028'	X'D8210003'	Error obtaining storage for RCVINFO block.
	X'00000028'	X'D8210004'	Error obtaining storage for ALLOC block.
	X'00000028'	X'D8210005'	Error obtaining storage for IC block.
	X'00000028'	X'D8210006'	Error obtaining storage for REORG block.
	X'00000028'	X'D8210007'	Error obtaining storage for RECOV block.
	X'00000028'	X'D9100001'	An error occurred processing the request. DBRC releases any storage obtained up to this point. However, another error was encountered during the attempt to release storage.
Internal error	X'0000002C'	X'D8000001'	Failure opening the RECON data set.
	X'0000002C'	X'D8210001'	Failure locating the first DBDS record.
	X'0000002C'	X'D8210002'	Failure locating the specified DBDS record.
	X'0000002C'	X'D8210003'	Failure locating the next DBDS record.
	X'0000002C'	X'D8210004'	Failure locating the first Area Auth record.
	X'0000002C'	X'D8210005'	Failure locating the first ALLOC record.
	X'0000002C'	X'D8210006'	Failure locating the next ALLOC record.
	X'0000002C'	X'D8210007'	Failure locating the first IC record.
	X'0000002C'	X'D8210008'	Failure locating the next IC record.
	X'0000002C'	X'D8210009'	Failure locating the first REORG record.
	X'0000002C'	X'D821000A'	Failure locating the next REORG record.
	X'0000002C'	X'D821000B'	Failure locating the first RECOV record.
	X'0000002C'	X'D821000C'	Failure locating the next RECOV record.
	X'0000002C'	X'D821000D'	Failure locating the specified group record (DBDS group).
	X'0000002C'	X'D821000E'	Failure locating the specified group record (CA group).
	X'0000002C'	X'D821000F'	Failure locating a DBDS from the specified group record.
	X'0000002C'	X'D8210010'	Failure locating the DB record with the specified DBNAME.
	X'0000002C'	X'D8210011'	Failure attempting to locate the first AVAIL IC record.
	X'0000002C'	X'D8210012'	Failure attempting to locate the next AVAIL IC record.

Table 89. DSPAPI FUNC=QUERY TYPE=DBDS query return and reason codes (continued)

Code type	Return code	Reason code	Meaning
Parameter error	X'00000030'	X'C9000001'	The function (FUNC) specified in the parameter list that is passed to the API is invalid.
	X'00000030'	X'C9000002'	Invalid TOKEN field address. The address of the field containing the API TOKEN failed validity checking. The address specifies storage that is not owned by the calling program.
	X'00000030'	X'C9000003'	Invalid RETCODE field address. The address of the field to contain the API RETCODE failed validity checking. The address specifies storage that is not owned by the calling program.
	X'00000030'	X'C9000004'	Invalid RSNCODE field address. The address of the field to contain the API RSNCODE failed validity checking. The address specifies storage that is not owned by the calling program.
	X'00000030'	X'C9000005'	Invalid OUTPUT field address. The address of the field to contain the OUTPUT address failed validity checking. The address specifies storage that is not owned by the calling program.
	X'00000030'	X'C9000008'	Invalid DBNAME or GROUP address. The address of the field containing the DBNAME or GROUP failed validity checking. The address specifies storage that is not owned by the calling program.
	X'00000030'	X'C9000009'	Invalid DDN address. The address of the field containing the DDN failed validity checking. The address specifies storage that is not owned by the calling program.
	X'00000030'	X'D8000001'	Missing or invalid OUTPUT parameter.
	X'00000030'	X'D8000002'	Invalid value specified for TYPE parameter.
	X'00000030'	X'D8000003'	Query TYPE=DBDS requires a minimum API version of 2.0.
	X'00000030'	X''D8210001'	The DBNAME or GROUP is required.
	X'00000030'	X'D8210002'	DDN specified with invalid LOC value. Only LOC=NEXT|SPEC can be entered.
	X'00000030'	X'D8210003'	Invalid LOC value. When no DDN is specified only LOC=FIRST can be used.
	X'00000030'	X'D8210004'	The DDN parameter is not allowed with GROUP parameter.
	X'00000030'	X'D8210005'	The LOC is not allowed with GROUP parameter.

The following block mappings relate to the TYPE=DB and TYPE=DBDS request:

- DSPAPQAR – Fast Path AREA block
- DSPAPQDS – DBDS block
- DSPAPQEL – EEQE List
- DSPAPQSL – Subsystem List (Fast Path only)

- DSPAPQRI – Recovery Information (RCVINFO) block
- DSPAPQAL – Allocation block
- DSPAPQIC – Image Copy block
- DSPAPQRV – Recovery block
- DSPAPQRR – Reorganization block

Related concepts

"DBRC API" on page 341
Your applications can obtain services from Database Recovery Control (DBRC) through the DBRC application programming interface (API), a release-independent, assembler macro interface. The application obtains these services by issuing DBRC API requests to DBRC, and DBRC returns the results to an area in storage where the application can retrieve them.

Related reference

"DBRC query request (QUERY)" on page 367
You can use the DBRC Query request (DSPAPI FUNC=QUERY) along with the TYPE parameter to retrieve various types of information from the RECON data set.

"Database query request (TYPE=DB)" on page 373
You can use the Database Query request (DSPAPI FUNC=QUERY TYPE=DB) to retrieve information from the RECON concerning one or more registered databases.

"COMMAND output block mapping" on page 364
These examples illustrate output block mapping for the DBRC command request. You can use the examples to understand how the DSPAPCMD output block header is structured and how the output appears.

Group query request (TYPE=*GROUP)

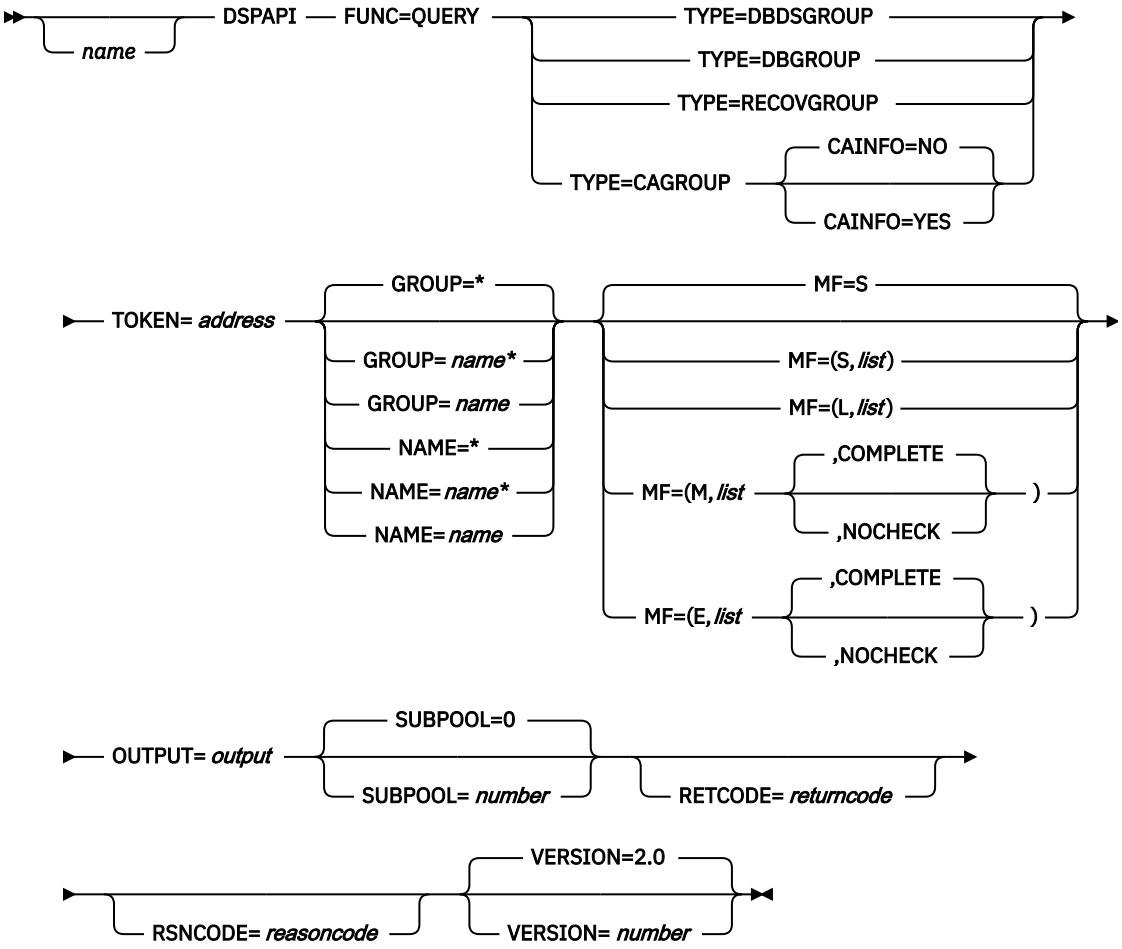
You can use the Group query (DSPAPI FUNC=QUERY TYPE=*GROUP) request to retrieve group and member information for various types of groups that are registered in the RECON data set.

The following list outlines the groups that you can retrieve information for through the Group query (DSPAPI FUNC=QUERY TYPE=*GROUP) request:

- DBDS group (TYPE=DBDSGROUP)
- DB group (TYPE=DBGROUP)
- Recovery group (TYPE=RECOVGROUP)
- CA group (TYPE=CAGROUP)

Subsections:

Syntax for the TYPE=*GROUP query request



Parameters for the TYPE=*GROUP query request

name
Optional symbol you can specify. If used, begins in column 1.

TYPE=DBDSGROUP | DBGROUP | RECOVGROUP | CAGROUP
Specifies the type of group for which information is requested.

CAINFO= YES | NO
Specifies whether CA execution information is to be included with the CAGROUP information. CAINFO is valid only with TYPE=CAGROUP. CAINFO defaults to NO when TYPE=CAGROUP is specified, indicating that only the CA group member information is requested.

If CAINFO=NO is specified or if CAINFO=YES is specified and no CA execution information exists, the block-dependent pointer (apqhd_depptr) in the header of the CAGROUP block is 0.

TOKEN=*symbol* | (2 - 12)
Specifies the address of a 4-byte field that was returned on the FUNC=STARTDBRC request.

GROUP= * | *symbol* | *symbol | (2 - 12)**
Specifies the name of the group being queried. You can use the wildcard keyword * (an asterisk) alone to request information about all groups. You can also use the wildcard at the end of a name, in which case the asterisk must be preceded by at least one alphabetic character. The default is GROUP=*.

Either the GROUP keyword or the NAME keyword can be used. GROUP is preferred. NAME is accepted for compatibility.

NAME= * | *symbol* | *symbol | (2 - 12)**
See the GROUP parameter.

MF=S | L | M | E
Specifies the macro form of the request.

OUTPUT=*output* | (2 - 12)
Specifies a field to receive a pointer to the first block of a possible chain of group information blocks. See "Output for the TYPE=*GROUP query request" on page 404 for a detailed description of the information blocks returned.

The output address is zero if no output was built which can occur if nothing in the RECON satisfies the request or if an error occurs before any output could be built.

The storage for the output blocks is not preallocated by the caller. DBRC acquires storage from the specified subpool for these blocks. The caller must free this storage using the Buffer Release service (DSPAPI FUNC=RELBUF) and specify the returned output address.

SUBPOOL= 0 | *number*
Specifies the subpool number for the storage being obtained. If not specified, the default is the subpool specified by the FUNC=STARTDBRC request. Otherwise, subpool 0 is the default.

RETCODE=*returncode* | (2-12)
If specified as a symbol, specifies the label of a word of storage to receive the return code. If specified as a register, the register must contain the address of a word of storage to receive the return code. Regardless of whether RETCODE is specified, register 15 contains the return code.

RSNCODE=*reasoncode* | (2-12)
If specified as a symbol, the symbol must be the label of a word of storage to receive the reason code. If specified as a register, the register must contain the address of a word of storage to receive the reason code. Regardless of whether RSNCODE is specified, register 0 contains the reason code.

VERSION=2.0 | *number*
Specifies the version number of the parameter list to be generated by this request.

To use the parameters associated with a version, you must specify the number of that version or a later version. If you specify an earlier version level, the parameter is not accepted by macro processing, and an error message is issued at assembly time. If parameters have a version dependency, the parameter descriptions with each request type identify the version number required.

Valid version numbers for the FUNC=QUERY TYPE=xxxxGROUP request are 1.0 and 2.0.

Return and reason codes for the TYPE=*GROUP query request

The following table contains most of the return and reason codes for TYPE=*GROUP query requests.

*Table 90. Return and reason codes for TYPE=*GROUP query requests*

Code type	Return codes	Reason codes	Meaning
	X'00000000'	X'00000000'	Request completed successfully.
Warning	X'00000008'	X'D8300001'	No group records of the requested type exist in the RECON data set.
Severe error	X'0000000C'	X'C9000001'	Invalid TOKEN. The TOKEN block passed to the API is not recognized as a TOKEN created by a FUNC=STARTDBRC call.
	X'0000000C'	X'C900000A'	The TCB address is not the same as the TCB address under which the STARTDBRC service was issued.
	X'0000000C'	X'D8300100'	Security error. SAF or the DBRC Command Authorization exit routine (DSPDCAX0) determined that the user is not authorized to perform the request.
Storage error	X'00000028'	X'D8300001'	Error obtaining storage for CAGROUP block.

*Table 90. Return and reason codes for TYPE=*GROUP query requests (continued)*

Code type	Return codes	Reason codes	Meaning
	X'00000028'	X'D8300003'	Error obtaining storage for DBDSGRP, DBGRP, or RECOVGRP block.
	X'00000028'	X'D8310001'	Error obtaining storage for CA block.
	X'00000028'	X'D9100001'	An error occurred processing the request. DBRC releases storage that was obtained up to this point. However, another error was encountered during the attempt to release storage.
Internal error	X'0000002C'	X'D8000001'	RECON open failure.
	X'0000002C'	X'D8300001'	Failure locating a specific group record or the first group record of the requested group type.
	X'0000002C'	X'D8300002'	Failure locating the next group record of the requested group type.
	X'0000002C'	X'D8300003'	Failure locating a CA record.
Parameter error	X'00000030'	X'C9000001'	The function (FUNC) specified in the parameter list passed to the API is invalid.
	X'00000030'	X'C9000002'	Invalid TOKEN field address. The address of the field containing the API TOKEN failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000003'	Invalid RETCODE field address. The address of the field to contain the API RETCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000004'	Invalid RSNCODE field address. The address of the field to contain the API RSNCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000005'	Invalid OUTPUT field address. The address of the field to contain the OUTPUT address failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000008'	Invalid GROUP or NAME field address. The address of the field containing the group name failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'D8000001'	Missing or invalid OUTPUT parameter.
	X'00000030'	X'D8000002'	Invalid value specified for TYPE parameter.
	X'00000030'	X'D8300001'	CAINFO=YES is only allowed with TYPE=CAGROUP.
	X'00000030'	X'D8300100'	When using a wildcard, at least one alphabetic character must precede the asterisk.
	X'00000030'	X'D8300101'	When using a wildcard, the asterisk must be the last character.

Output for the TYPE=*GROUP query request

The next few figures illustrate the format of the output from a QUERY TYPE=*GROUP requests. Following the figures that graphically describe the layout of the output are sample DSECTs that describe in detail the fields of the storage blocks and their relationship to each other.

Output for QUERY TYPE=DBDSGROUP, DBGROUP, and RECOVGROUP

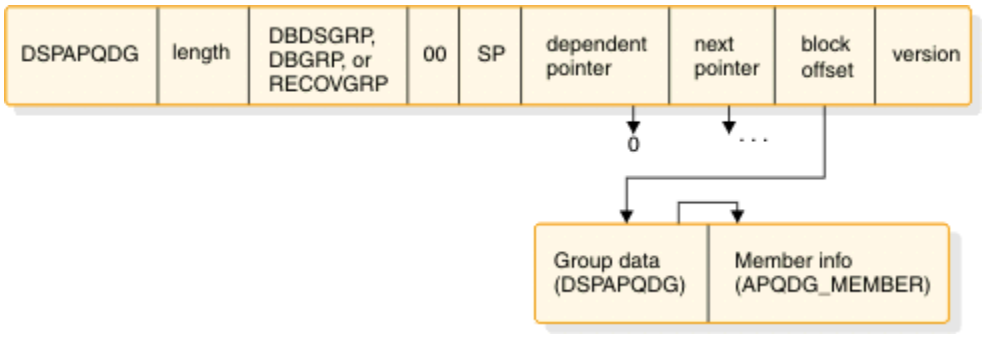


Figure 20. Format for QUERY TYPE=DBDSGROUP, DBGROUP, RECOVGROUP output

DSECT of DSPAPQDG

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE         LENGTH  NAME (DIM)                          DESCRIPTION
======== ======== ========= ========  ==============                      ============================
       0      (0) STRUCTURE        32  DSPAPQDG
       0      (0) CHARACTER         8   APQDG_GROUPNAME                        Group name
       8      (8) UNSIGNED          4   APQDG_MEMBERINFO          Offset to group member list
      12      (C) UNSIGNED          2   APQDG_MEMBERLEN           Length of group member entry
      14      (E) SIGNED            2   APQDG_MEMBERCOUNT         Number of group members
      16     (10) SIGNED            4   *(3)                      Reserved
      28     (1C) CHARACTER         4   *                         Reserved

  OFFSET   OFFSET
 DECIMAL     HEX  TYPE         LENGTH  NAME (DIM)                          DESCRIPTION
======== ======== ========= ========  ==============                      ============================
       0      (0) STRUCTURE        16  APQDG_MEMBER                        List of group members
       0      (0) CHARACTER        16   *
       0      (0) CHARACTER        16    APQDG_DBDSG              DBDS group
       0      (0) CHARACTER         8     APQDG_DBDSG_DBNAME     Database name
       8      (8) CHARACTER         8     *
       8      (8) CHARACTER         8      APQDG_DBDSG_DDNAME    DD name or
       8      (8) CHARACTER         8      APQDG_DBDSG_AREANAME  AREA name
       0      (0) CHARACTER        16    APQDG_DBG                DB group
       0      (0) CHARACTER         8     *
       0      (0) CHARACTER         8      APQDG_DBG_DBNAME      Database name or
       0      (0) CHARACTER         8      APQDG_DBG_AREANAME    AREA name
       8      (8) CHARACTER         8     *                       Not used
       0      (0) CHARACTER        16    APQDG_RECOVG             Recovery group
       0      (0) CHARACTER         8     APQDG_RECOVG_DBNAME    Database name
       8      (8) CHARACTER         8     APQDG_RECOVG_AREANAME  AREA name, null if
                                                                   not Fast Path

CONSTANTS

  LEN  TYPE           VALUE           NAME            DESCRIPTION
=====  =========  ================  ============== ================================
    8  CHARACTER   DSPAPQDG          APQDG_EYECATCHER
```

Output for QUERY TYPE=CAGROUP

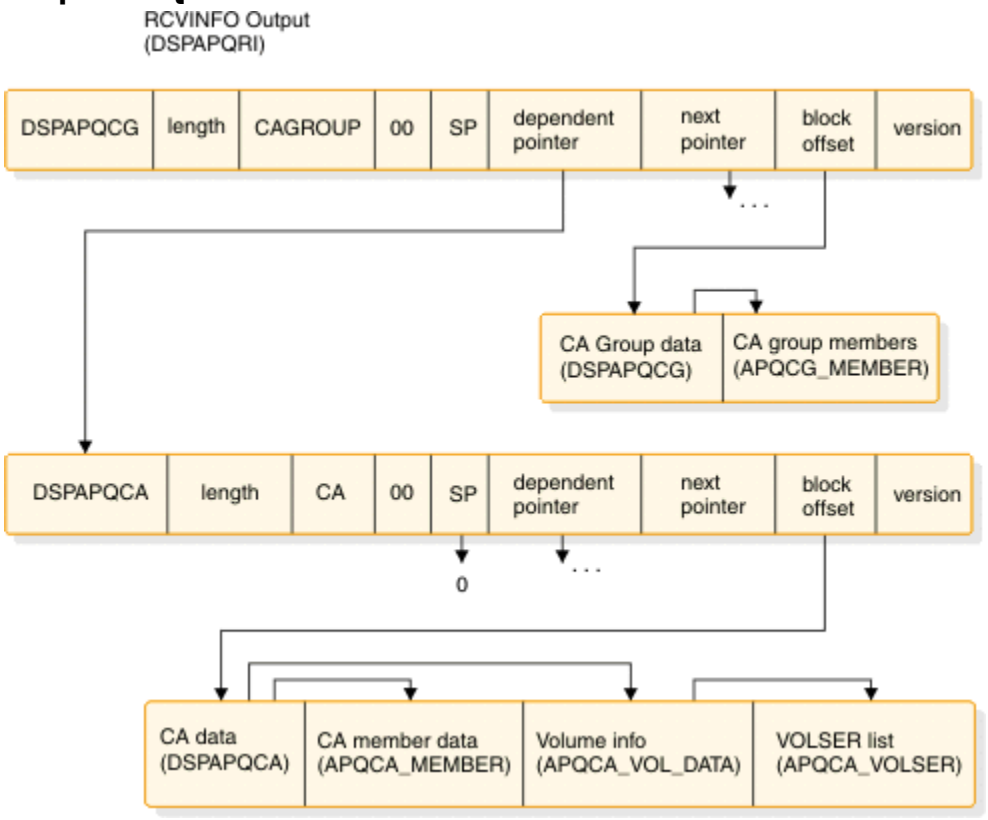


Figure 21. Format for QUERY TYPE=CAGROUP output

The CA block is only returned when CAINFO=YES is specified and records of a change accumulation exist in the RECON.

DSECT of DSPAPQCG

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)                DESCRIPTION
======== ======== ========= ========  ==============            ================================
       0      (0) STRUCTURE       48  DSPAPQCG
       0      (0) CHARACTER        8   APQCG_GROUPNAME        Group name
       8      (8) UNSIGNED         4   APQCG_MEMBERINFO        Offset to group member list
      12      (C) UNSIGNED         2   APQCG_MEMBERLEN        Length of group member entry
      14      (E) SIGNED           2   APQCG_MEMBERCOUNT  Number of group members
      16     (10) SIGNED           4   *(2)                      Reserved
      24     (18) SIGNED           2   APQCG_GRPMAX              Maximum number of CAs that may
                                                              be predefined for this CA group
      26     (1A) SIGNED           2   APQCG_AVAILCA#            Number of available CA data
                                                              sets for this group
      28     (1C) SIGNED           2   APQCG_USEDCA#             Number of used CA data sets
      30     (1E) BIT(8)           1   APQCG_FLAGS               Flags
                  1... ....             APQCG_REUSE               Reuse CA data sets
      31     (1F) CHARACTER        1   *                         Reserved
      32     (20) CHARACTER        8   APQCG_CAJCL               GENJCL CAJCL member name
      40     (28) CHARACTER        8   APQCG_DFJCL               DEFLTJCL member name
      48        (30) SIGNED           2  APQCG_RECOVPD       Retention Period (Version=4.0)

  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)                DESCRIPTION
======== ======== ========= ========  ==============            ================================
       0      (0) STRUCTURE       16  APQCG_MEMBER              List of group members
       0      (0) CHARACTER        8   APQCG_DBNAME              Database name
       8      (8) CHARACTER        8   *
       8      (8) CHARACTER        8    APQCG_DDNAME             DD name or
       8      (8) CHARACTER        8    APQCG_AREANAME        AREA name

CONSTANTS

  LEN  TYPE          VALUE          NAME                      DESCRIPTION
```

```
=====  =========  =================  ===============            ==============================
    8  CHARACTER   DSPAPQCG          APQCG_EYECATCHER
```

DSECT of DSPAPQCA

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH   NAME (DIM)             DESCRIPTION
======== ======== ========= ========   ==============         ==============================
       0      (0) STRUCTURE        96   DSPAPQCA
       0      (0) CHARACTER         8   APQCA_GROUPNAME        Group name
       8      (8) UNSIGNED          4   APQCA_MEMBERINFO        Offset to group member list
      12      (C) UNSIGNED          2   APQCA_MEMBERLEN        Length of each member entry
      14      (E) SIGNED            2   APQCA_MEMBERCOUNT    Number of group members
      16     (10) UNSIGNED          4   APQCA_VOLINFO            Offset to volume information
      20     (14) SIGNED            4   *                        Reserved
      24     (18) CHARACTER        44   APQCA_DSN                Data set name
      68     (44) CHARACTER        12   APQCA_STOPTIME           Packed decimal date/time - for
                                                                   predefined datasets, represents
                                                                   record creation time.
                                                                   Otherwise, it is the stoptime
                                                                   of the last logtape volume used
                                                                   as input to the Change
                                                                   Accumulation utility that
                                                                   produced this CA as output. If
                                                                   the CA run included an
                                                                   'incomplete log subset' it is
                                                                   the start time of the first
                                                                   truncated log volume.
      80     (50) CHARACTER        12   APQCA_RUNTIME            CA run time
      92     (5C) BIT(8)            1   APQCA_FLAGS              Flags
                  1... ....              APQCA_ERROR              Error on data set
                  .1.. ....              APQCA_SUBSET             Subset of logs used for CA
                  ..1. ....              APQCA_COMMAND            SUBSET/COMP has been set or
                                                                   reset with an external cmd
                  ...1 ....              APQCA_AVAIL      Available CA indicator
      93     (5D) CHARACTER         3   *                        Reserved
      96     (60) UNSIGNED          4   APQCA_OFFUD              Offset to user data (VERSION=4.0)
     100     (64) UNSIGNED          2   APQCA_LENUD              Length of user data (VERSION=4.0)

   OFFSET   OFFSET
  DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)          DESCRIPTION
 ======== ======== ========= ========  ==============      ==============================
        0      (0) STRUCTURE        80  APQIC_USER          Data for user IC
        0      (0) CHARACTER        80  APQRV_UDATA          User supplied data (VERSION=4.0)

  OFFSET  OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)                    DESCRIPTION
======== ======== ========= ========  ==============                ================================
       0      (0) STRUCTURE        56  APQCA_MEMBER                List of group members
       0      (0) CHARACTER         8   APQCA_MEM_DBNAME          Database name
       8      (8) CHARACTER         8   *
       8      (8) CHARACTER         8    APQCA_MEM_DDNAME         DD name or
       8      (8) CHARACTER         8    APQCA_MEM_AREANAME       AREA name
      16     (10) UNSIGNED          4   APQCA_MEM_DSSN            Data Set sequence number
      20     (14) UNSIGNED          4   APQCA_MEM_USID            USID of last change
                                                                   accumulated
      24     (18) CHARACTER         8   APQCA_MEM_LRID            LRID of last change accumulated
      32     (20) CHARACTER        12   APQCA_MEM_PURGETIME       Purge time
      44     (2C) CHARACTER         6   APQCA_MEM_LSN             Lock sequence number
      50     (32) BIT(8)            1   APQCA_MEM_FLAGS           Member flags
                  1... ....              APQCA_MEM_NOCHG          No changes accumulated
                  .1.. ....              APQCA_MEM_INDOUBT        Indoubt EEQEs accumulated
                  ..1. ....              APQCA_MEM_INCOMP         Incomplete CA
      51     (33) CHARACTER         5   *                        Reserved
```

Related concepts

“Macro forms of the DSPAPI macro” on page 344

There are four different macro forms for the DSPAPI macro: Standard (S), List (L), Modify (M), and Execute (E), with two variations, COMPLETE and NOCHECK. The List, Modify, and Execute forms are usually used in combinations when writing reentrant programs or when the application issues multiple requests.

Related reference

“DBRC query request (QUERY)” on page 367

You can use the DBRC Query request (DSPAPI FUNC=QUERY) along with the TYPE parameter to retrieve various types of information from the RECON data set.

DBRC request return codes (Messages and Codes)

Log query request (TYPE=LOG)

You can use the Log query (DSPAPI FUNC=QUERY TYPE=LOG) request to retrieve log information from RECON for a specific instance of a subsystem.

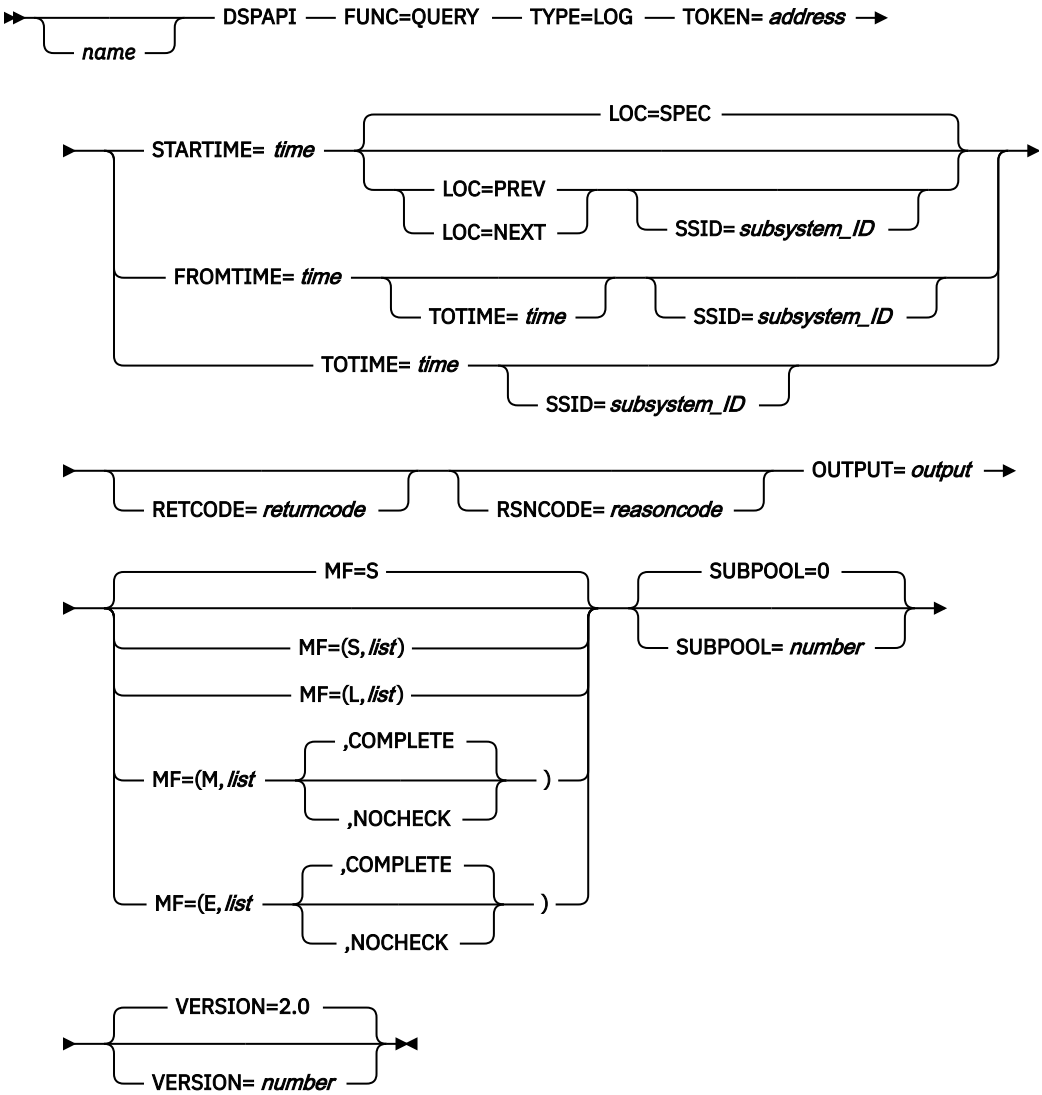
Information from the following RECON data sets is returned:

- PRILOG
- LOGALL
- SECLOG (if applicable)
- PRISLDS (if applicable)
- SECSLDS (if applicable)

The log query request can also return log information for subsystems that started in a specified time range. This request can also be used for a specific subsystem.

Subsections:

Syntax for the TYPE=LOG query request



Parameters for the TYPE=LOG query request

name
Optional symbol you can specify. If used, begins in column 1.

TYPE=LOG
Specifies that log information is requested.

TOKEN=*address* | (2 - 12)
Specifies the address of a 4-byte field that was returned on the FUNC=STARTDBRC request.

STARTIME=*time* | (2-12)
Specifies the time stamp field that contains the start time of the requested log. The time is a packed decimal time stamp in UTC format.

FROMTIME=*time* | (2-12)
Specifies the time stamp field that limits the logs requested to those whose subsystem started at, or after, the specified time. The time is a packed decimal time stamp in UTC format.

If you specify this parameter, you must also specify a minimum version number of DBRC API VERSION=2.0.

TOTIME=*time* | (2-12)
Specifies the time stamp field that limits the logs requested to those whose subsystem started at, or before, the specified time. The time is a packed decimal time stamp in UTC format. This parameter may be used along with the FROMTIME parameter.

If you specify this parameter requires that you also specify a minimum version number of API VERSION=2.0.

SSID=*subsystem_ID* | (2-12)
Specifies the subsystem name for the log being queried.

You can specify the SSID parameter only when requesting the previous or next log of a specific subsystem, for example, when LOC previous or next is specified, or a range of logs for a specific subsystem using the FROMTIME or TOTIME parameters.

LOC=PREV | NEXT | SPEC
Specifies that the request is for the log with a specified start time (LOC=SPEC), a start time preceding the specified start time (LOC=PREV), or a start time following the specified start time (LOC=NEXT). The STARTIME parameter is used as the base of the search and does not need to be the start time of a login RECON.

LOC=SPEC is the default.

MF=S | L | M | E
Specifies the macro form of the request.

OUTPUT=*output* | (2 - 12)
Specifies a field to receive a pointer to the first block of a possible chain of log information blocks.

The output address is zero if no output was built. This can occur if nothing in the RECON satisfies the request or if an error occurs before any output could be built.

The storage for the output blocks is not pre-allocated by the caller. DBRC acquires storage from the specified subpool for these blocks. The caller must free this storage using the Buffer Release service (DSPAPI FUNC=RELBUF) and specify the returned output address.

SUBPOOL= 0 | *number*
Specifies the subpool number for the storage being obtained. If not specified, the default is the subpool specified by the FUNC=STARTDBRC request. Otherwise, subpool 0 is the default.

RETCODE=*returncode* | (2-12)
If specified as a symbol, specifies the label of a word of storage to receive the return code. If specified as a register, the register must contain the address of a word of storage to receive the return code. Regardless of whether RETCODE is specified, register 15 contains the return code.

RSNCODE=*reasoncode* | (2-12)
If specified as a symbol, the symbol must be the label of a word of storage to receive the reason code. If specified as a register, the register must contain the address of a word of storage to receive the reason code. Regardless of whether RSNCODE is specified, register 0 contains the reason code.

VERSION=2.0|*number*
Specifies the version number of the parameter list that is generated by this macro.

To use the parameters associated with a version, you must specify the number of that version or a later version. If you specify an earlier version level, the parameter is not accepted by macro processing, and an error message is issued at assembly time. If parameters have a version dependency, the parameter descriptions with each request type identify the version number required.

Valid version numbers for the FUNC=QUERY TYPE=LOG request are 1.0 and 2.0.

Return and reason codes for the TYPE=LOG query request

Table 91. Return and reason codes for TYPE=LOG query requests

Code type	Return codes	Reason codes	Meaning
	X'00000000'	X'00000000'	Request completed successfully.
Warning	X'00000008'	X'D8400001'	No log record of the requested log type (PRILOG) exists. The request was the previous or next log or logs within a time range specified by FROMTIME or TOTIME.
	X'00000008'	X'D8400002'	The specified log record of the requested log type - PRILOG - does not exist.
Severe error	X'0000000C'	X'C9000001'	Invalid TOKEN. The TOKEN block passed to the API is not recognized as a TOKEN created by a FUNC=STARTDBRC call.
	X'0000000C'	X'C900000A'	The TCB address is not the same as the TCB address under which the STARTDBRC service was issued.
	X'0000000C'	X'D8400100'	Security error. SAF or the DBRC Command Authorization exit routine (DSPDCAX0) determined that the user is not authorized to perform the request.
Storage error	X'00000028'	X'D8400001'	Error obtaining storage for LOGINFO block.
	X'00000028'	X'D8400002'	Error obtaining storage for PRILOG, SECLOG, PRISLDS, or SECSLDS block.
	X'00000028'	X'D8400003'	Error obtaining storage for LOGALL block.
	X'00000028'	X'D9100001'	An error occurred processing the request. DBRC will release storage that was obtained up to this point. However, another error was encountered during the attempt to release storage.
Internal error	X'0000002C'	X'D8000001'	RECON open failure.
	X'0000002C'	X'D8400001'	Failure locating the previous or next log record of the requested log type - PRILOG.
	X'0000002C'	X'D8400002'	Failure locating the specified log record of the requested log type - PRILOG.
	X'0000002C'	X'D8400004'	Failure locating the LOGALL record that corresponds to the PRILOG record.
	X'0000002C'	X'D8400005'	The LOGALL record that corresponds to the PRILOG record does not exist.
	X'0000002C'	X'D8400006'	Failure locating the corresponding SECLOG record.
	X'0000002C'	X'D8400007'	Failure locating the corresponding PRISLDS record.
	X'0000002C'	X'D8400008'	No PRISLDS record exists for the online log.
	X'0000002C'	X'D8400009'	Failure locating the corresponding SECSLDS record.
	X'0000002C'	X'D840000A'	Failure occurred in DBRC Time Services processing FROMTIME.
Parameter error	X'00000030'	X'C9000001'	The function (FUNC) specified in the parameter list passed to the API is invalid.

Table 91. Return and reason codes for TYPE=LOG query requests (continued)

Code type	Return codes	Reason codes	Meaning
	X'00000030'	X'C9000002'	Invalid TOKEN field address. The address of the field containing the API TOKEN failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000003'	Invalid RETCODE field address. The address of the field to contain the API RETCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000004'	Invalid RSNCODE field address. The address of the field to contain the API RSNCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000005'	Invalid OUTPUT field address. The address of the field to contain the OUTPUT address failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000010'	Invalid SSID field address. The address of the field containing the SSID failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000011'	Invalid STARTIME field address. The address of the field containing the STARTIME failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000012'	Invalid FROMTIME field address. The address of the field that contains the FROMTIME parameter failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000013'	Invalid TOTIME field address. The address of the field that contains the TOTIME parameter failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'D8000001'	Missing or invalid OUTPUT parameter.
	X'00000030'	X'D8000002'	Invalid value specified for TYPE parameter.
	X'00000030'	X'D8400001'	The STARTIME, FROMTIME, or TOTIME parameter is required.
	X'00000030'	X'D8400002'	SSID is required with TRACKER=YES.
	X'00000030'	X'D8400003'	The SSID parameter is not allowed when querying a specific active log (LOC=SPEC and TRACKER=NO).
	X'00000030'	X'D8400006'	The FROMTIME parameter value must be less than the TOTIME parameter value.
	X'00000030'	X'D8400007'	STARTIME cannot be specified with FROMTIME | TOTIME.
	X'00000030'	X'D8400008'	LOC cannot be specified with FROMTIME or TOTIME.

Table 91. Return and reason codes for TYPE=LOG query requests (continued)

Code type	Return codes	Reason codes	Meaning
	X'00000030'	X'D8400010'	The value passed in the FROMTIME parameter is not a valid time

Output for the TYPE=LOG query request

The following figures illustrate the format of output from a QUERY TYPE=LOG requests. The sample DSECTs that follow the figures describe in detail the fields of the storage blocks and their relationship to each other.

Log information output

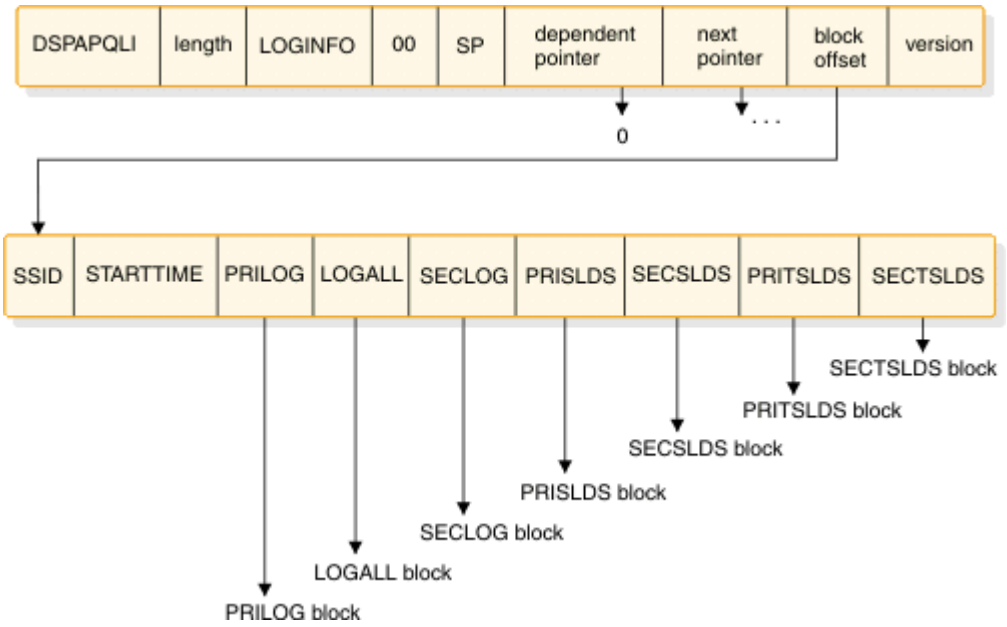


Figure 22. Format for QUERY TYPE=LOG log information output

DSECT of DSPAPQLI

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)         DESCRIPTION
======== ======== ========= ========  ==============     ================================
       0      (0) STRUCTURE       48  DSPAPQLI
       0      (0) CHARACTER        8   APQLI_SSID        Log SSID
       8      (8) CHARACTER       12   APQLI_STARTTIME   Log start time
      20     (14) ADDRESS          4   APQLI_PRILOGPTR   ptr to PRILOG block
      24     (18) ADDRESS          4   APQLI_LOGALLPTR   ptr to LOGALL block
      28     (1C) ADDRESS          4   APQLI_SECLOGPTR   ptr to SECLOG block
      32     (20) ADDRESS          4   APQLI_PRISLDSPTR  ptr to PRISLDS block
      36     (24) ADDRESS          4   APQLI_SECSLDSPTR  ptr to SECSLDS block
      40     (28) ADDRESS          4   APQLI_PRITSLDSPTR ptr to PRITSLDS block
      44     (2C) ADDRESS          4   APQLI_SECTSLDSPTR ptr to SECTSLDS block

CONSTANTS

  LEN  TYPE           VALUE           NAME            DESCRIPTION
=====  =========  ================  ============== ================================
    8  CHARACTER   DSPAPQLI          APQLI_EYECATCHER
```

TYPE=LOG output for PRILOG, SECLOG, PRISLDS, and SECSLDS

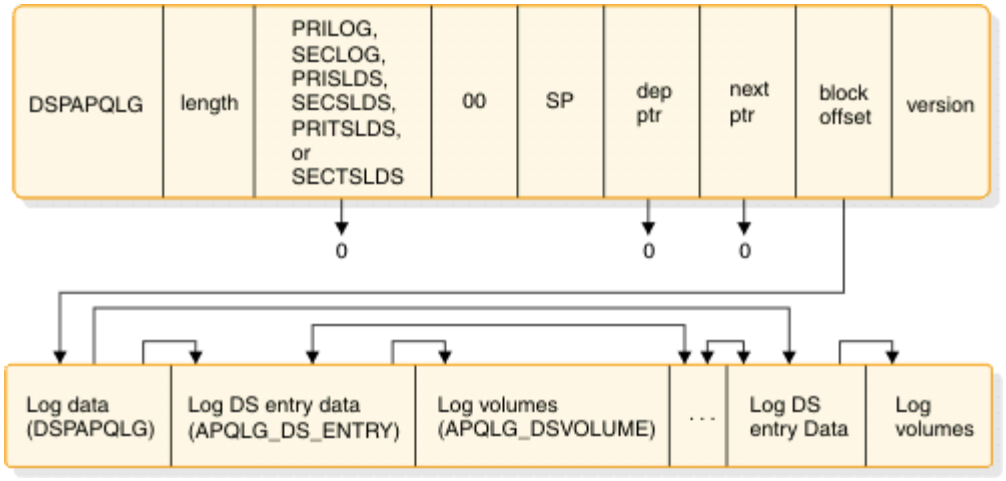


Figure 23. Format for QUERY TYPE=LOG output for PRILOG, SECLOG, PRISLDS, and SECSLDS

DSECT of DSPAPQLG

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)         DESCRIPTION
======== ======== ========= ========  ==============     ================================
       0      (0) STRUCTURE       96  DSPAPQLG
       0      (0) UNSIGNED         4   APQLG_FIRSTLOGDS  Offset to first log DS entry
       4      (4) UNSIGNED         4   APQLG_LASTLOGDS   Offset to last log DS entry
       8      (8) SIGNED           4   *(2)              Reserved
      16     (10) CHARACTER        8   APQLG_SSID        Subsystem ID
      24     (18) CHARACTER       12   APQLG_STARTTIME   Log start time
      36     (24) CHARACTER       12   APQLG_ENDTIME     Log end time
      48     (30) SIGNED           4   APQLG_DSNCOUNT    Number of data sets
      52     (34) UNSIGNED         1   APQLG_RELVL       Log Release Level
                  1... ....             APQLG_ONLINE      Online log - PRILOG and SECLOG only
                  .1.. ....             APQLG_LSTAR       Last OLDS has been archived - PRILOG and
PRITSLDS only
                  ..1. ....             APQLG_LSTNA       Last OLDS has not been archived - PRILOG and
PRITSLDS only
                  ...1 ....             APQLG_GAP         There is a gap in this log
                  .... 1...             APQLG_PRGAP       There is a gap in a prev log
      53     (34) UNSIGNED         1   APQLG_FLAGS1      Flags
                  1... ....             APQLG_ONLINE      Online log - PRILOG and SECLOG only
                  .1.. ....             APQLG_LSTAR       Last OLDS has been archived – PRILOG
                                                          and PRITSLDS only
                  ..1. ....             APQLG_LSTNA       Last OLDS has not been archived –
                                                          PRILOG and PRITSLDS only
                  ...1 ....             APQLG_GAP         There is a gap in this log
                  .... 1...             APQLG_PRGAP       There is a gap in a prev log
                  .... .1..             APQLG_BPE         BPE-based subsystem (VERSION=4.0)
      54     (36) BIT(8)           1   APQLG_FLAGS2      Flags
                  1... ....             APQLG_TRKNG       Tracking log data set
                  .1.. ....             APQLG_NTERM       IMS subsystem has terminated normally
                  ..1. ....             APQLG_BKLOG       Batch backout log
      55     (37) CHARACTER        1   *                 Reserved
      56     (38) CHARACTER        8   APQLG_FIRSTLRID   Id of first log record
      64     (40) UNSIGNED         4   APQLG_PTOKEN      PRILOG token
      68     (44) CHARACTER        8   APQLG_GSGNAME     GSG name
      76     (4C) CHARACTER       12   APQLG_CHKPT0      Checkpoint 0 time
      88     (58) CHARACTER        8   *                 Reserved  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)             DESCRIPTION
======== ======== ========= ========  ==============         ================================
       0      (0) STRUCTURE      120  APQLG_DS_ENTRY         Data set entry
       0      (0) UNSIGNED         4   APQLG_DS_NEXT         Offset to next log DS entry
       4      (4) UNSIGNED         4   APQLG_DS_PREV         Offset to prev log DS entry
       8      (8) UNSIGNED         4   APQLG_DS_VOLINFO      Offset to DS volume info
      12      (C) CHARACTER       44   APQLG_DS_DSNAME       Data set name
      56     (38) CHARACTER       12   APQLG_DS_STARTTIME    DS start time
      68     (44) CHARACTER       12   APQLG_DS_ENDTIME      DS end time
      80     (50) BIT(8)           1   APQLG_DS_FLAGS1       Flags
                  1... ....             APQLG_DS_ERR I/O      Error
                  .1.. ....             APQLG_DS_DUMMY        Log compressed, 1st DS dummy
                  ..1. ....             APQLG_DS_RSTBG        Restart begin
```

```
                  ...1 ....              APQLG_DS_RSTEN       Restart end
                  .... 1...              APQLG_DS_COLD        Cold start
                  .... .1..              APQLG_DS_NOBMP       ERE NOBMP
                  .... ..1.              APQLG_DS_SAVER       Backout UORs saved
                  .... ...1              APQLG_DS_NOID        Backouts not identified
      81     (51) BIT(8)            1   APQLG_DS_FLAGS2      Flags
                  1... ....               APQLG_DS_TRKARCH    Tracking log DS archived
                  .1.. ....               APQLG_DS_TRKFEOV    Tracking log closed FEOV
      82     (52) CHARACTER         2   APQLG_DS_DFLG3       Reserved
      84     (54) CHARACTER         8   APQLG_DS_FLRID       First log record ID
      92     (5C) CHARACTER         8   APQLG_DS_LLRID       Last log record ID
     100     (64) UNSIGNED          4   APQLG_DS_LASTBLKSEQNO Last block seq number
     104     (68) CHARACTER         8   APQLG_DS_UNITTYPE    Unit type name
     112     (70) UNSIGNED          2   APQLG_DS_FILESEQ     File sequence no
     114     (72) UNSIGNED          2   APQLG_DS_VOLCOUNT    Number of volumes
     116     (74) UNSIGNED          1   APQLG_DS_CKPTCOUNT   Number of chkpts on DSN
     117     (75) BIT(8)            1   APQLG_DS_CHKPTTYPES  CHKPT types.
                  1... ....               APQLG_DS_SIMPL      SIMPLE checkpoint
                  .1.. ....               APQLG_DS_SNAPQ      SNAPQ checkpoint
                  ..1. ....               APQLG_DS_DUMPQ      DUMPQ checkpoint
                  ...1 ....               APQLG_DS_PURGE      PURGE checkpoint
                  .... 1...               APQLG_DS_FREEZ      FREEZE checkpoint
     118     (76) UNSIGNED          2   *                    Reserved
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE         LENGTH  NAME (DIM)            DESCRIPTION
======== ======== ========= ========  ============          ================================
       0      (0) STRUCTURE        48  APQLG_DSVOLUME        Volume information
       0      (0) UNSIGNED          4   APQLG_DSVOL_NEXT     Offset to next volume
       4      (4) CHARACTER         6   APQLG_DSVOL_SER      VOLSER
      10      (A) UNSIGNED          1   APQLG_DSVOL_CKPTCT   Volume chkpt count
      11      (B) CHARACTER         1   *                    Reserved
      12      (C) CHARACTER        12   APQLG_DSVOL_ENDTIME  Volume end time
      24     (18) CHARACTER        12   APQLG_DSVOL_CPTID    Checkpoint ID
      36     (24) CHARACTER         6   APQLG_DSVOL_LOCKSN   Lock Sequence Number
      42     (2A) CHARACTER         6   *                    Reserved

CONSTANTS

  LEN  TYPE           VALUE           NAME                DESCRIPTION
=====  =========  ================  ==============       ================================
    8  CHARACTER   DSPAPQLG          APQLG_EYECATCHER
```

TYPE=LOG output for LOGALL

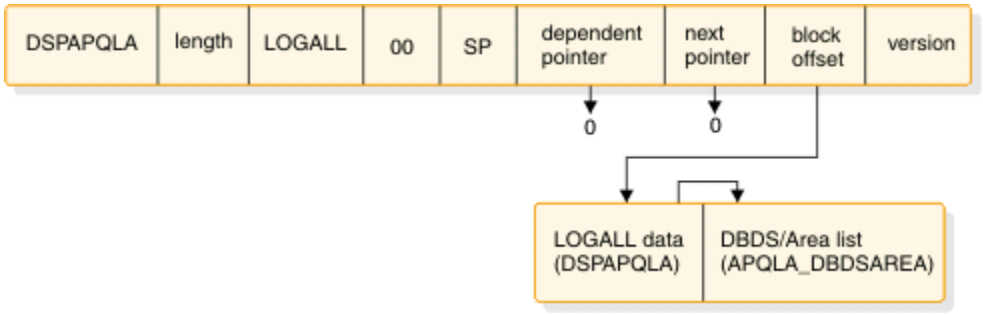


Figure 24. Format for QUERY TYPE=LOG output for LOGALL

DSECT of DSPAPQLA

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE         LENGTH  NAME (DIM             DESCRIPTION
======== ======== ========= ========  ==============        ================================
       0      (0) STRUCTURE        48  DSPAPQLA
       0      (0) UNSIGNED          4   APQLA_DBDSAREAINFO       Offset to allocated
                                                              DBDS/Area list
       4      (4) SIGNED            4   *(3)                  Reserved
      16     (10) CHARACTER        12   APQLA_PRILOGTIME      PRILOG time
      28     (1C) BIT(8)            1   APQLA_FLAGS           Flags
                  1... ....              APQLA_NONREGD        Non-registered DB updated
      29     (1D) UNSIGNED          3   APQLA_DBDSAREACOUNT   Number of DBDS/Areas
                                                              allocated on this log
      32     (20) UNSIGNED          4   APQLA_DBDSAREALEN     Length of DBDS/Area entry
      36     (24) CHARACTER        12   APQLA_EARLIESTALLOC   Earliest ALLOC time for
                                                              this log
```

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)            DESCRIPTION
======== ======== ========= ========  ==============        ================================
       0      (0) STRUCTURE       32  APQLA_DBDSAREA        List of allocated DBDSs and
                                                            areas
       0      (0) CHARACTER        8   APQLA_DBNAME         Database name
       8      (8) CHARACTER        8   *
       8      (8) CHARACTER        8    APQLA_DDNAME        DD name or
       8      (8) CHARACTER        8    APQLA_AREANAME      AREA name
      16     (10) CHARACTER       12   APQLA_FIRSTALLOC     Earliest ALLOC time for this
                                                            DBDS/Area on this log
      28     (1C) SIGNED           2   APQLA_ALLNO          Number of ALLOCs for this
                                                            DBDS/Area on this log
      30     (1E) CHARACTER        2   *                    Reserved

CONSTANTS

  LEN  TYPE           VALUE           NAME                  DESCRIPTION
=====  =========  ================  ==============        ================================
    8  CHARACTER   DSPAPQLA          APQLA_EYECATCHER
```

Related concepts

“Macro forms of the DSPAPI macro” on page 344

There are four different macro forms for the DSPAPI macro: Standard (S), List (L), Modify (M), and Execute (E), with two variations, COMPLETE and NOCHECK. The List, Modify, and Execute forms are usually used in combinations when writing reentrant programs or when the application issues multiple requests.

Related reference

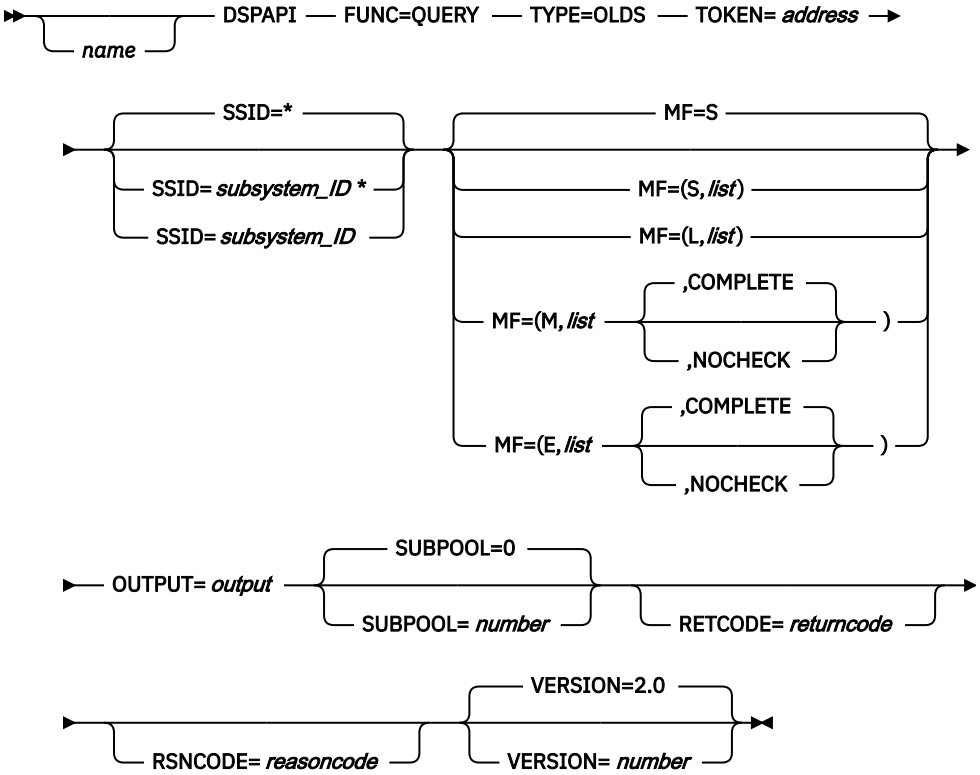
“DBRC query request (QUERY)” on page 367

You can use the DBRC Query request (DSPAPI FUNC=QUERY) along with the TYPE parameter to retrieve various types of information from the RECON data set.

OLDS query request (TYPE=OLDS)

You can use the OLDS query (DSPAPI FUNC=QUERY TYPE=OLDS) request to retrieve online log data set information from the RECON for a specific subsystem or all subsystems.

Syntax for the TYPE=OLDS query request



Parameters for the TYPE=OLDS query request

name
Optional symbol you can specify. If used, begins in column 1.

TYPE=OLDS
Specifies that online log data set information is requested.

TOKEN=*symbol* | (2 - 12)
Specifies the address of a 4-byte field that was returned on the FUNC=STARTDBRC request.

SSID=* | *symbol | *symbol* | (2 - 12)**
Specifies the subsystem name for the backout being queried. You can use the wildcard keyword * (an asterisk) alone to request all of the OLDS information (SSID=*, the default). You can also use it at the end of a name to query subsystem names that match the pattern. In this case, the asterisk must be preceded by at least one alphabetic character.

MF=S | L | M | E
Specifies the macro form of the request.

OUTPUT=*output* | (2 - 12)
Specifies a field to receive a pointer to the first block of a possible chain of OLDS information blocks.

The output address is zero if no output was built. This can occur if nothing in the RECON satisfies the request or if an error occurs before any output could be built.

The storage for the output blocks is not pre-allocated by the caller. DBRC acquires storage from the specified subpool for these blocks. The caller must free this storage using the Buffer Release service (DSPAPI FUNC=RELBUF) and specify the returned output address.

SUBPOOL= 0 | *number*
Specifies the subpool number for the storage being obtained. If not specified, the default is the subpool specified by the FUNC=STARTDBRC request. Otherwise, subpool 0 is the default.

RETCODE=*returncode* | (2-12)
If specified as a symbol, specifies the label of a word of storage to receive the return code. If specified as a register, the register must contain the address of a word of storage to receive the return code. Regardless of whether RETCODE is specified, register 15 contains the return code.

RSNCODE=*reasoncode* | (2-12)
If specified as a symbol, the symbol must be the label of a word of storage to receive the reason code. If specified as a register, the register must contain the address of a word of storage to receive the reason code. Regardless of whether RSNCODE is specified, register 0 contains the reason code.

VERSION=2.0 | *number*
Specifies the version number of the parameter list to be generated by this request.

To use the parameters associated with a version, you must specify the number of that version or a later version. If you specify an earlier version level, the parameter is not accepted by macro processing, and an error message is issued at assembly time. If parameters have a version dependency, the parameter descriptions with each request type identify the version number required.

Valid version numbers for the FUNC=QUERY TYPE=OLDS request are 1.0 and 2.0 (default).

Return and reason codes for the TYPE=OLDS query request

Table 92. Return and reason codes for TYPE=OLDS query requests

Code type	Return codes	Reason codes	Meaning
	X'00000000'	X'00000000'	Request completed successfully.
Warning	X'00000008'	X'D8500001'	No PRIOLDS records exist.
Severe error	X'0000000C'	X'C9000001'	Invalid TOKEN. The TOKEN block passed to the API is not recognized as a TOKEN created by a FUNC=STARTDBRC call.
	X'0000000C'	X'C900000A'	The TCB address is not the same as the TCB address under which the STARTDBRC service was issued.
	X'0000000C'	X'D8500100'	Security error. SAF or the DBRC Command Authorization exit routine (DSPDCAX0) determined that the user is not authorized to perform the request.
Storage error	X'00000028'	X'D8500001'	Error obtaining storage for OLDS block.
	X'00000028'	X'D9100001'	An error occurred processing the request. DBRC will release storage that was obtained up to this point. However, another error was encountered during the attempt to release storage.
Internal error	X'0000002C'	X'D8000001'	RECON open failure.
	X'0000002C'	X'D8500001'	Failure locating the first or the specified PRIOLDS record.
	X'0000002C'	X'D8500002'	Failure locating the corresponding SECOLDS record.
	X'0000002C'	X'D8500003'	Failure locating the next PRIOLDS record.

Table 92. Return and reason codes for TYPE=OLDS query requests (continued)

Code type	Return codes	Reason codes	Meaning
Parameter error	X'00000030'	X'C9000001'	The function (FUNC) specified in the parameter list passed to the API is invalid.
	X'00000030'	X'C9000002'	Invalid TOKEN field address. The address of the field containing the API TOKEN failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000003'	Invalid RETCODE field address. The address of the field to contain the API RETCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000004'	Invalid RSNCODE field address. The address of the field to contain the API RSNCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000005'	Invalid OUTPUT field address. The address of the field to contain the OUTPUT address failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000010'	Invalid SSID field address. The address of the field containing the SSID failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'D8000001'	Missing or invalid OUTPUT parameter.
	X'00000030'	X'D8500100'	When using a wildcard, at least one alphabetic character must precede the asterisk.
	X'00000030'	X'D8500101'	When using a wildcard, the asterisk must be the last character.

Output for the TYPE=OLDS query request

The following figure illustrates the format of the output from a QUERY TYPE=OLDS request. The following DSECT sample describes in detail the fields of the storage blocks and their relationship to each other.

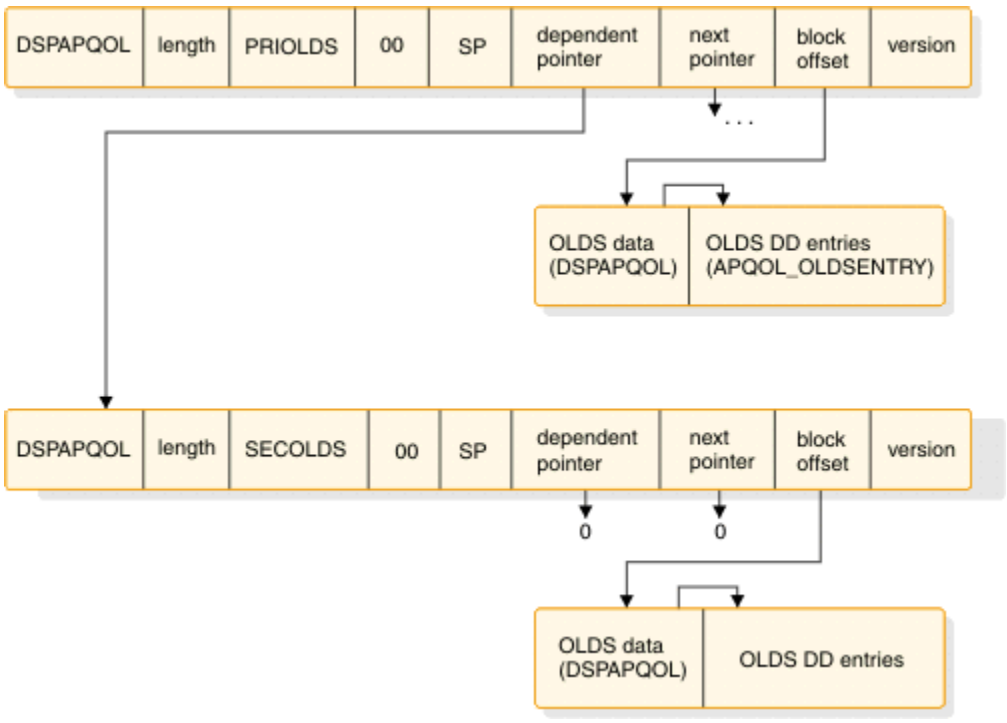


Figure 25. Format for QUERY TYPE=OLDS output

DSECT of DSPAPQOL

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)       DESCRIPTION
======== ======== ========= ========  ==============   ================================
       0      (0) STRUCTURE       48  DSPAPQOL
       0      (0) UNSIGNED         4   APQOL_OLDSINFO  Offset to OLDS list
       4      (4) SIGNED           4   *(3)            Reserved
      16     (10) CHARACTER        8   APQOL_SSID      Subsystem ID
      24     (18) UNSIGNED         2   APQOL_OLDSLEN   Length of OLDS entry
      26     (1A) SIGNED           2   APQOL_OLDSCOUNT Number of OLDS entries
      28     (1C) CHARACTER       12   APQOL_CHKPT0    Checkpoint 0 time
      40     (28) CHARACTER        8   *               Reserved

  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)        DESCRIPTION
======== ======== ========= ========  ==============    ================================
       0      (0) STRUCTURE      128  APQOL_OLDSENTRY   OLDS entry
       0      (0) CHARACTER        8   APQOL_DDNAME     OLDS DD name
       8      (8) CHARACTER       44   APQOL_DSNAM      OLDS data set name
      52     (34) CHARACTER       12   APQOL_OPENTIME   OLDS open time
      64     (40) CHARACTER       12   APQOL_CLOSETIME  Close time
      76     (4C) CHARACTER       12   APQOL_PRILOGTIME Start time of associated
                                                        PRILOG
      88     (58) CHARACTER        8   APQOL_FLSN       LSN of first record in OLDS
      96     (60) CHARACTER        8   APQOL_LLSN       LSN of last record in OLDS
     104     (68) BIT(8)           1   APQOL_FLAG1
                  1... ....             APQOL_RSTBG     Restart begin
                  .1.. ....             APQOL_RSTEN     Restart end
                  ..1. ....             APQOL_COLD      Cold start
                  ...1 ....             APQOL_NOBMP     ERE NOBMP
                  .... 1...             APQOL_SAVER     Backout UORs saved
                  .... .1..             APQOL_NOID      Backouts not identified
                  .... ..1.             APQOL_TRKNG     OLDS created by tracking SS
     105     (69) BIT(8)           1   APQOL_FLAG2      OLDS flags
                  1111 ....             APQOL_STAT      OLDS status
                  1... ....              APQOL_INUSE    OLDS is in use
                  .1.. ....              APQOL_ARNED    Archive needed
                  ..1. ....              APQOL_ARSCH    Archive scheduled (GENJCL)
                  ...1 ....              APQOL_ARSTD    Archive job started
                  .... 1...             APQOL_CLERR     Close error on OLDS
                  .... .1..             APQOL_FEOV      Force EOV at archive
                  .... ..1.             APQOL_DUMMY     OLDS not used due to I/O err
                  .... ...1             APQOL_PRVCE     Close error on previous OLDS
     106     (6A) UNSIGNED         1   APQOL_RELVL      Log release level
```

```
     107     (6B) UNSIGNED         1   APQOL_GAVER      GENJCL.ARCHIVE version
     108     (6C) BIT(32)          4   APQOL_BLOCKSEQNO Block sequence number
     112     (70) CHARACTER        8   APQOL_ARJOB      Name of the archive job
                                                        Generated by GENJCL.ARCHIVE
     120     (78) CHARACTER        6   APQOL_LOCKSEQNO  Lock sequence number
     126     (7E) CHARACTER        2   *                Reserved

CONSTANTS

  LEN  TYPE           VALUE           NAME             DESCRIPTION
=====  =========  ================  ================ ================================
    8  CHARACTER   DSPAPQOL          APQOL_EYECATCHER
```

Related concepts

“Macro forms of the DSPAPI macro” on page 344
There are four different macro forms for the DSPAPI macro: Standard (S), List (L), Modify (M), and Execute (E), with two variations, COMPLETE and NOCHECK. The List, Modify, and Execute forms are usually used in combinations when writing reentrant programs or when the application issues multiple requests.

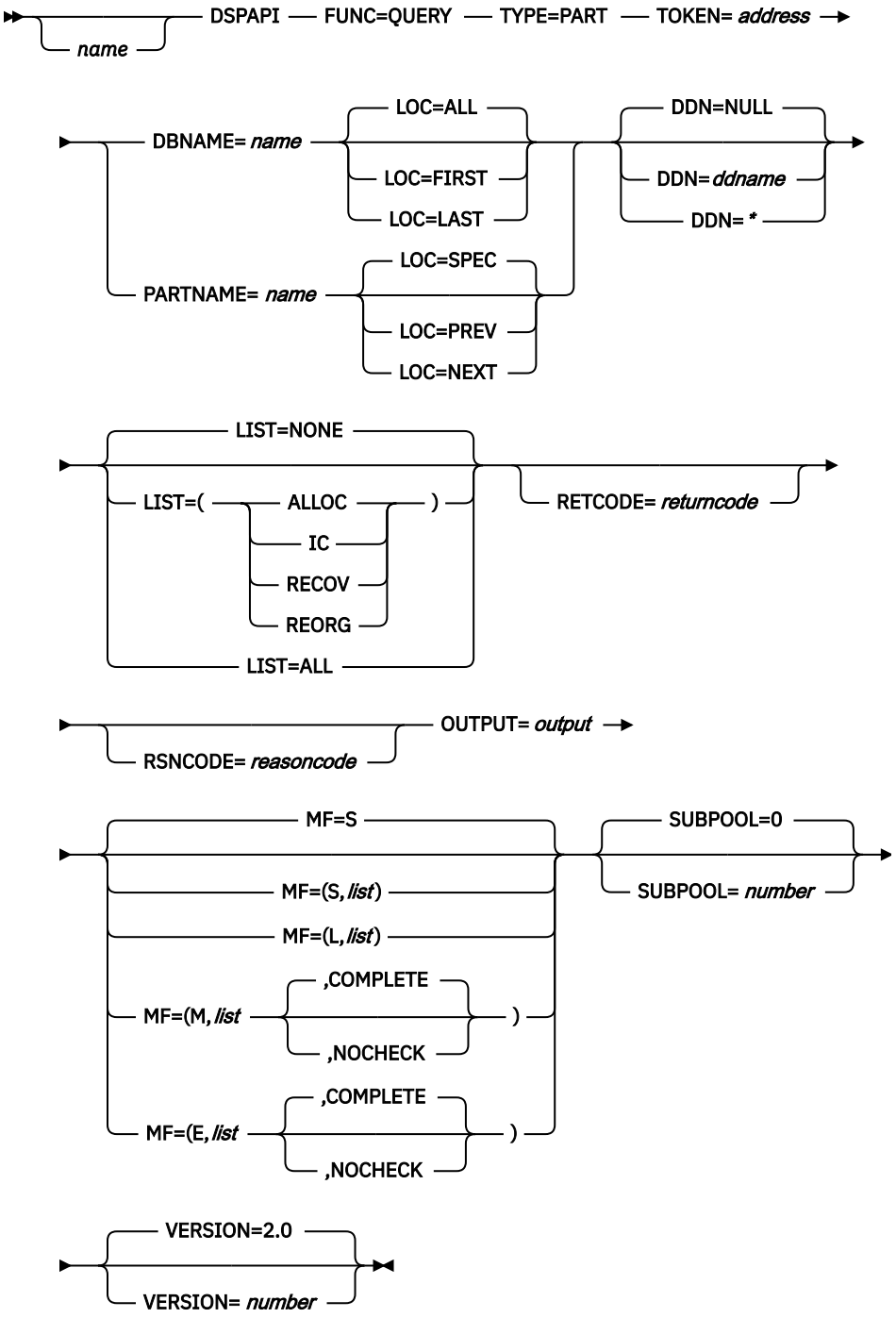
Related reference

“DBRC query request (QUERY)” on page 367
You can use the DBRC Query request (DSPAPI FUNC=QUERY) along with the TYPE parameter to retrieve various types of information from the RECON data set.

HALDB partition query request (TYPE=PART)

You can use the DSPAPI FUNC=QUERY TYPE=PART request to retrieve information for a particular HALDB partition from the RECON data set. You can request data set information for a specific DBDS or all DBDSs in the partition, and can optionally request recovery-related information for the data set, including allocation, image copy, recovery, and reorganization information.

Syntax for the TYPE=PART query request



Parameters for the TYPE=PART query request

name
Optional symbol you can specify. If used, begins in column 1.

TYPE=PART
Required parameter that specifies that RECON data set information for a HALDB partition is requested.

When you specify the TYPE=PART, you must also specify a minimum DBRC API version number of VERSION=2.0.

DBNAME=*name* | (2 - 12)
Specifies the HALDB name of the partition being queried. You can optionally specify the LOC parameter, which indicates that you are interested in either the first partition, last partition, or all partitions of the specified HALDB. If the HALDB uses high keys, the request is for the partition with the lowest or highest high key. Otherwise, the first or last alphanumerical partition is returned. LOC=SPEC|PREV|NEXT may not be specified.

PARTNAME=*name* | (2 - 12)
Required parameter that specifies the name of a HALDB partition being queried.

The LOC parameter indicates the partition of interest:

- The default LOC=SPEC indicates a request for the specified partition. This partition might be available or disabled.
- LOC=PREV indicates a request for the active partition in the HALDB that is before the specified partition. If the HALDB uses high keys, the request is for the partition with the lower high key. Otherwise, the previous alphanumerical partition within the HALDB is returned. The partition name specified must be the name of an active partition registered in the RECON. This is required in order to ensure that the previous partition is within the same HALDB
- LOC=NEXT indicates a request for the active partition in the HALDB that is after the specified partition. If the HALDB uses high keys, the request is for the partition with the higher high key. Otherwise, the next alphanumerical partition within the HALDB is returned. The partition name specified must be the name of an active partition registered in the RECON. This is required to ensure that the next partition is within the same HALDB.
- LOC=FIRST|LAST can not be specified with PARTNAME.

To use this parameter, you must also specify a minimum version number of API VERSION=2.0.

DDN=NULL | (2 - 12) | * / *ddname*
Specifies the DD name of a DBDS set within the partition. An asterisk (* without quotes) can also be specified indicating that information about all DBDSs is requested. If a specific DDN is requested and the request is not for a specific partition information is returned for all DBDSs of the partition.

DDN=NULL indicates that no DBDS information is requested. This is the default.

LIST=({ALLOC},{IC},{RECOV},{REORG}) | LIST=ALL | LIST=NONE
Specifies the types of supporting information to be included in the query output for the specified DBDS. If DDN is not specified, this information is returned for all DBDSs in the partition. One or more of the specific values is included in the list: ALLOC (allocation records), IC (image copy records), RECOV (recovery records), or REORG (reorganization records). LIST=ALL requests all supporting information. LIST=NONE requests no supporting information.

LOC=ALL | FIRST | LAST | PREV | NEXT | SPEC

The optional parameter specifies that the request is for the specified partition (SPEC), or for the first, last, previous, next or all active partitions defined in RECON for the HALDB. LOC=SPEC may return an active or disabled partition. DBNAME is required with LOC=FIRST|LAST|ALL, but is not allowed with LOC=PREV|NEXT|SPEC. Conversely, PARTNAME is required with LOC=PREV|NEXT|SPEC, but is not allowed with LOC=FIRST|LAST|ALL. LOC=ALL is the default with DBNAME. LOC=SPEC is the default with PARTNAME.

Partitions are returned in high key order if the HALDB uses high keys. Otherwise, partitions are returned in alphanumerical order.

TOKEN=*address* | (2 - 12)
Specifies the address of a 4-byte field to receive the API token. This token must be included in all subsequent requests associated with this FUNC=STARTDBRC request.

RETCODE=*returncode* | (2 - 12)
Specifies a location in storage to receive the return code. If specified as a symbol, the symbol must be the label of a word of storage. If specified as a register, the register must contain the address of a word of storage. If not specified, the return code is placed in register 15.

RSNCODE=*reasoncode* | (2 - 12)
Specifies a place in storage to receive the reason code. If specified as a symbol, the symbol must be the label of a word of storage. If specified as a register, the register must contain the address of a word of storage. If not specified, the reason code is placed in register 0.

OUTPUT=*output* | (2 - 12)
Required parameter that specifies a field to receive a pointer to the first of a possible chain of blocks that contain the information for the partition.

The output address is 0 if no output is built. This can happen if nothing in the RECON data set satisfies the request or if an error occurs before any output is built.

The storage for the output blocks is not pre-allocated by the caller. DBRC acquires storage from the specified subpool for these blocks. The caller must free this storage using the Buffer Release service (DSPAPI FUNC=RELBUF) and specify the returned output address.

SUBPOOL= 0 | *number*
Optional parameter that specifies the subpool number for the storage being obtained. If not specified, the default is the subpool specified by the FUNC=STARTDBRC request. Otherwise, subpool 0 is the default.

MF=S | L | M | E
Specifies the macro form of the request.

VERSION=2.0 | *number*
Optional parameter that specifies the version number of the parameter list that is generated by this request.

To use the parameters associated with a version, you must specify the number of that version or a later version. If you specify an earlier version level, the parameter is not accepted for processing and an error message is issued at assembly time. If parameters have a version dependency, the parameter descriptions with each request type identify the version number required.

The default version is 2.0.

Note: TYPE=PART requires that you specify a minimum version number of API VERSION=2.0.

Return and reason codes for the TYPE=PART query request

The following table contains most of the return and reason codes for TYPE=PART query requests. The other possible return and reason codes relate to DBRC, not the query request.

Table 93. Return and reason codes for TYPE=PART query requests

Code type	Return codes	Reason codes	Meaning
	X'00000000'	X'00000000'	Request completed successfully.
Warning	X'00000008'	X'D8210002'	The specified DBDS or Area is not registered in RECON. No information blocks are returned.
	X'00000008'	X'D8220001'	No partitions are registered in RECON for the HALDB. No information blocks are returned.
	X'00000008'	X'D8220002'	The specified partition is not registered in RECON. No information blocks are returned.
	X'00000008'	X'D8220003'	A high key partition preceding the specified partition does not exist in RECON. No information blocks are returned.
	X'00000008'	X'D8220004'	A high key partition following the specified partition does not exist in RECON. No information blocks are returned.

Table 93. Return and reason codes for TYPE=PART query requests (continued)

Code type	Return codes	Reason codes	Meaning
	X'00000008'	X'D8220005'	The DBNAME specified is not registered in RECON. No information blocks are returned.
	X'00000008'	X'D8220006'	No active partitions found while searching for the FIRST or LAST partition of the HALDB.
	X'00000008'	X'D8220007'	No active partitions found while searching for the PREV or NEXT partition.
Severe error	X'0000000C'	X'C9000001'	Invalid TOKEN. The TOKEN block passed to the API is not recognized as a TOKEN created by a FUNC=STARTDBRC call.
	X'0000000C'	X'C900000A'	The TCB address is not the same as the TCB address under which the STARTDBRC service was issued.
	X'0000000C'	X'D8220100'	Security error. SAF or the DBRC cmd auth exit (DSPDCAX0) has determined that the user is not authorized to perform the request.
Storage error	X'00000028'	X'D8210001'	Error obtaining storage for DBDS block.
	X'00000028'	X'D8210002'	Error obtaining storage for AREA block.
	X'00000028'	X'D8210003'	Error obtaining storage for RCVINFO block.
	X'00000028'	X'D8210004'	Error obtaining storage for ALLOC block.
	X'00000028'	X'D8210005'	Error obtaining storage for IC block.
	X'00000028'	X'D8210006'	Error obtaining storage for REORG block.
	X'00000028'	X'D8210007'	Error obtaining storage for RECOV block.
	X'00000028'	X'D8220001'	Error obtaining storage for PART block.
	X'00000028'	X'D9100001'	An error occurred processing the request. DBRC will release storage that was obtained up to this point. However, another error was encountered during the attempt to release storage.
Internal error	X'0000002C'	X'D8000001'	RECON open failure.
	X'0000002C'	X'D8210001'	Failure locating the first DBDS record.
	X'0000002C'	X'D8210002'	Failure locating the specified DBDS record.
	X'0000002C'	X'D8210003'	Failure locating the next DBDS record.
	X'0000002C'	X'D8210004'	Failure locating the first Area Auth record.
	X'0000002C'	X'D8210005'	Failure locating the first ALLOC record.
	X'0000002C'	X'D8210006'	Failure locating the next ALLOC record.
	X'0000002C'	X'D8210007'	Failure locating the first IC record.
	X'0000002C'	X'D8210008'	Failure locating the next IC record.
	X'0000002C'	X'D8210009'	Failure locating the first REORG record.
	X'0000002C'	X'D821000A'	Failure locating the next REORG record.
	X'0000002C'	X'D821000B'	Failure locating the first RECOV record.

Table 93. Return and reason codes for TYPE=PART query requests (continued)

Code type	Return codes	Reason codes	Meaning
	X'0000002C'	X'D821000C'	Failure locating the next RECOV record.
	X'0000002C'	X'D8220001'	Failure locating the first HALDB partition record.
	X'0000002C'	X'D8220002'	Failure locating the DB record associated with the HALDB partition being processed.
	X'0000002C'	X'D8220003'	Failure locating the next HALDB partition record.
	X'0000002C'	X'D8220004'	Failure locating the specified HALDB (DBNAME).
	X'0000002C'	X'D8220005'	Failure locating the first or last active partition.
	X'0000002C'	X'D8220006'	Failure locating the first or last high key partition.
	X'0000002C'	X'D8220007'	Failure locating the previous or next high key partition.
	X'0000002C'	X'D8220008'	Failure locating the PART record corresponding to an existing Part DB record.
	X'0000002C'	X'D8220009'	Failure locating the previous or next partition of HALDB that uses a part selection routine.
	X'0000002C'	X'D822000A'	Failure locating the HALDB record for the specified partition.
	X'0000002C'	X'D822000B'	Failure locating the Part DB record for the specified PARTNAME.
	X'0000002C'	X'D822000C'	Neither the PART record nor the Part DB record is passed to ProcessPART.
	X'0000002C'	X'D822000D'	Failure locating the PART record associated with the HALDB partition being processed.
	X'0000002C'	X'D82221xx'	Internal Query DBDS call returned RC=X'30' RSN=X'D82100xx', a parameter error.
Parameter error	X'00000030'	X'C9000001'	The function (FUNC) specified in the parameter list passed to the API is invalid.
	X'00000030'	X'C9000002'	Invalid TOKEN field address. The address of the field containing the API TOKEN failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000003'	Invalid RETCODE field address. The address of the field to contain the API RETCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000004'	Invalid RSNCODE field address. The address of the field to contain the API RSNCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000005'	Invalid OUTPUT field address. The address of the field to contain the OUTPUT address failed validity checking. The address specifies storage not owned by the calling program.

Table 93. Return and reason codes for TYPE=PART query requests (continued)

Code type	Return codes	Reason codes	Meaning
	X'00000030'	X'C9000008'	Invalid DBNAME or PARTNAME address. The address of the field containing the DBNAME or PARTNAME failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000009'	Invalid DDN address. The address of the field containing the DDN failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'D8000001'	Missing or invalid OUTPUT parameter.
	X'00000030'	X'D8000002'	Invalid value specified for TYPE parameter.
	X'00000030'	X'D8000003'	TYPE=PART requires at least VERSION=2.0.
	X'00000030'	X'D8220002'	LOC=FIRST|LAST|ALL cannot be used with PARTNAME.
	X'00000030'	X'D8220003'	LOC=PREV|NEXT cannot be used with DBNAME.
	X'00000030'	X'D8220004'	Explicit name must be specified in DBNAME or PARTNAME. An asterisk cannot be used.
	X'00000030'	X'D8220005'	DBNAME specified is not a HALDB.
	X'00000030'	X'D8220006'	The partition name specified with LOC=PREV|NEXT must be an active partition.
	X'00000030'	X'D8220007'	PARTNAME must specify the name of a partition.

Output for TYPE=PART queries

The following mappings are used for the TYPE=PART query:

- DSPAPQHP - HALDB Partition block
- DSPAPQSL - Subsystem List

Related concepts

"DBRC API" on page 341
Your applications can obtain services from Database Recovery Control (DBRC) through the DBRC application programming interface (API), a release-independent, assembler macro interface. The application obtains these services by issuing DBRC API requests to DBRC, and DBRC returns the results to an area in storage where the application can retrieve them.

Related reference

"DBRC query request (QUERY)" on page 367
You can use the DBRC Query request (DSPAPI FUNC=QUERY) along with the TYPE parameter to retrieve various types of information from the RECON data set.

"Database query request (TYPE=DB)" on page 373
You can use the Database Query request (DSPAPI FUNC=QUERY TYPE=DB) to retrieve information from the RECON concerning one or more registered databases.

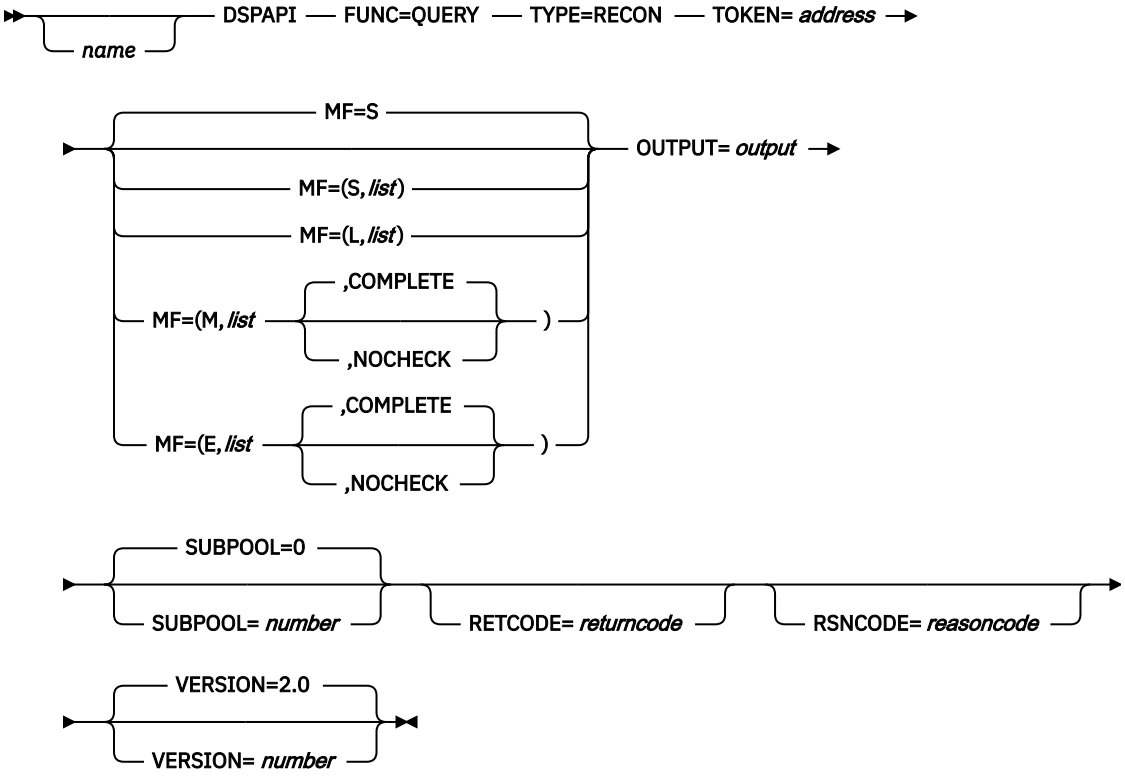
"DBDS query request (TYPE=DBDS)" on page 394
You can use the DSPAPI FUNC=QUERY TYPE=DBDS request to retrieve information from the RECON data set for one or more DBDSs in a non-HALDB database, a HALDB partition, a DBDS group, or a CA group. You can also request recovery related information for the data set, including allocation, image copy, recovery, and reorganization information.

DBRC request return codes (Messages and Codes)

RECON status query request (TYPE=RECON)

You can use the RECON status query (DSPAPI FUNC=QUERY TYPE=RECON) request to retrieve information pertaining to the RECON data sets, including RECON header information as well as the status of each RECON data set.

Syntax for the TYPE=RECON query request



Parameters for the TYPE=RECON query request

name
Optional symbol you can specify. If used, begins in column 1.

TYPE=RECON
Specifies that RECON status information is requested.

TOKEN=*symbol* | (2 - 12)
Required parameter that specifies the address of a 4-byte field that was returned on the FUNC=STARTDBRC request.

MF=S | L | M | E
Optional parameter that specifies the macro form of the request.

OUTPUT=*output* | (2 - 12)
Specifies a field to receive a pointer to the RECON status information block.

The output address is zero if no output was built. This can occur if nothing in the RECON satisfies the request or if an error occurs before any output could be built.

The storage for the output blocks is not preallocated by the caller. DBRC acquires storage from the specified subpool for these blocks. The caller must free this storage using the Buffer Release service (DSPAPI FUNC=RELBUF) and specify the returned output address.

SUBPOOL= 0 | *number*
Optional parameter that specifies the subpool number for the storage being obtained. If not specified, the default is the subpool specified by the FUNC=STARTDBRC request. Otherwise, subpool 0 is the default.

RETCODE=*returncode* | (2-12)
If specified as a symbol, specifies the label of a word of storage to receive the return code. If specified as a register, the register must contain the address of a word of storage to receive the return code. Regardless of whether RETCODE is specified, register 15 contains the return code.

RSNCODE=*reasoncode* | (2-12)
If specified as a symbol, the symbol must be the label of a word of storage to receive the reason code. If specified as a register, the register must contain the address of a word of storage to receive the reason code. Regardless of whether RSNCODE is specified, register 0 contains the reason code.

VERSION=2.0 | *number*
Specifies the version number of the parameter list to be generated by this request.

To use the parameters associated with a version, you must specify the number of that version or a later version. If you specify an earlier version level, the parameter is not accepted by macro processing, and an error message is issued at assembly time. If parameters have a version dependency, the parameter descriptions with each request type identify the version number required.

Valid version numbers are 1.0 and 2.0.

Return and reason codes for the TYPE=RECON query request

Table 94. Return and reason codes for TYPE=RECON queries

Code type	Return codes	Reason codes	Meaning
	X'00000000'	X'00000000'	Request completed successfully.
Severe error	X'0000000C'	X'C9000001'	Invalid TOKEN. The TOKEN block passed to the API is not recognized as a TOKEN created by a FUNC=STARTDBRC call.
	X'0000000C'	X'C900000A'	The TCB address is not the same as the TCB address under which the STARTDBRC service was issued.
	X'0000000C'	X'D8100100'	Security error. SAF or the DBRC Command Authorization exit routine (DSPDCAX0) determined that the user is not authorized to perform the request.
Storage error	X'00000028'	X'D8100001'	Error obtaining storage for the RECON block.
Internal error	X'0000002C'	X'D8000001'	RECON open failure.
	X'0000002C'	X'D8100001'	Failure attempting to locate the RECON header record.
Parameter error	X'00000030'	X'C9000001'	The function (FUNC) specified in the parameter list passed to the API is invalid.
	X'00000030'	X'C9000002'	Invalid TOKEN field address. The address of the field containing the API TOKEN failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000003'	Invalid RETCODE field address. The address of the field to contain the API RETCODE failed validity checking. The address specifies storage not owned by the calling program.

Table 94. Return and reason codes for TYPE=RECON queries (continued)

Code type	Return codes	Reason codes	Meaning
	X'00000030'	X'C9000004'	Invalid RSNCODE field address. The address of the field to contain the API RSNCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000005'	Invalid OUTPUT field address. The address of the field to contain the OUTPUT address failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'D8000001'	Missing or invalid OUTPUT parameter.
	X'00000030'	X'D8000002'	Invalid value specified for TYPE parameter.

Output for the TYPE=RECON query request

The following figure illustrates the format of the output from a QUERY TYPE=RECON request. The following DSECT sample describes in detail the fields of the storage blocks and their relationship to each other.

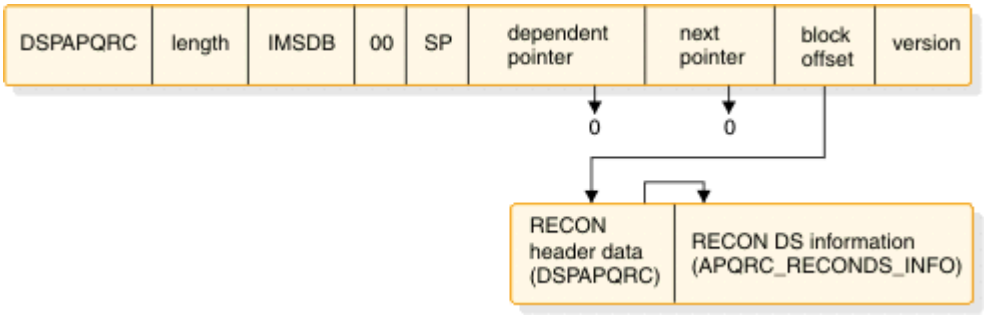


Figure 26. Format for QUERY TYPE=RECON output

DSECT of DSPAPQRC

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE         LENGTH  NAME (DIM)         DESCRIPTION
======== ======== ========= ========  ==============     ================================
       0      (0) STRUCTURE       560  DSPAPQRC
       0      (0) CHARACTER        44   APQRC_DATA        Initialized with "RECOVERY
                                                          CONTROL DATASET"
      44     (2C) UNSIGNED          4   APQRC_RECONINFO   Offset to RECON data set
                                                          info
      48     (30) SIGNED            4   *(2)              Reserved
      56     (38) UNSIGNED          2   APQRC_RECONINFOLEN  Length of each RECON dataset info element
      58     (3A) UNSIGNED          1   APQRC_RECONCOUNT    # of RECON dataset elements
      59     (3B) CHARACTER         1   *                 Reserved
      60     (3C) BIT(8)            1   APQRC_FLAGS       Process flags...
                  1... ....              APQRC_NOCHK       1= NOCHECK log tape dsn check
                  .1.. ....              APQRC_CHK17       1= CHECK17 log tape dsn check
                  ..1. ....              APQRC_CHK44       1= CHECK44 log tape dsn check
                  ...1 ....              APQRC_LSTLG       1= list log DSN
                  .... 1...              APQRC_INUPG       Upgrade in progress.
                  .... .1..              APQRC_REORGV      Reorg verification
      61     (3D) BIT(8)            1   APQRC_FLAG2       More flags...
                  1... ....              APQRC_FORCE       1 = FORCER, 0 = NOFORCER
                  .1.. ....              APQRC_CATDS       1=CA|IC|LOGS cataloged
                  ..1. ....              APQRC_TRACE       1 = ext. GTF trace on
                  ...1 ....              APQRC_CMDSAF      SAF enabled
                  .... 1...              APQRC_CMDEXIT     Cmd auth exit enabled
                  .... .1..              APQRC_PRA            Parallel RECON Access in use (VERSION=2.0)
                  .... ..1.              APQRC_LISTFUZZY   PRA Concurrent LIST active (VERSION=2.0)
      62     (3E) CHARACTER         2   *                 Reserved
      64     (40) CHARACTER       136   APQRC_CLEAN       Fields needed for cleanup
      64     (40) SIGNED            4    APQRC_CSET       0 = updates not in progress,
```

```
                                                    >0 = update in progress
     68     (44) SIGNED           4    APQRC_TYPE      Type of update
     72     (48) CHARACTER       32    APQRC_OKEY      Key of original record
    104     (68) CHARACTER       32    APQRC_BKEY      Key of base record that is in
                                                       the process of being changed
    136     (88) CHARACTER       32    APQRC_NKEY      Key of new record
    168     (A8) CHARACTER       16    APQRC_DBID      DBID of DBDS
    168     (A8) CHARACTER        8     APQRC_DBD      DBD name
    176     (B0) CHARACTER        8     APQRC_DDN      DD name
    184     (B8) CHARACTER        8    APQRC_CAGRP     CA group name
    192     (C0) CHARACTER        8    APQRC_DDNEW     New DBDS DD name
    200     (C8) UNSIGNED         2   APQRC_DMBNO      DMB sequence number
    202     (CA) UNSIGNED         2   APQRC_LASTREUSEDDMB#   Last reused DMB number, valid
                                                       only when apqrc_DMBNO is 32767
    204     (CC) CHARACTER        7   APQRC_INITTOKEN  Recon init. token
    211     (D3) CHARACTER        8   APQRC_CMDHLQ     Cmd auth high lvl qual
    219     (DB) UNSIGNED         1   APQRC_MVERS      Minimum IMS version
    220     (DC) BIT(8)           1   APQRC_NWFLG      Fields needed for RECON I/O
                                                       error(s)
```

```
                1... ....              APQRC_NEW    1=STARTNEW, 0=NONEW
    221     (DD) CHARACTER        3   *                Reserved
    224     (E0) CHARACTER        8   APQRC_SSIDN      SSID for DASD
    232     (E8) CHARACTER        8   APQRC_DASDU      Unit type for DASD
    240     (F0) CHARACTER        8   APQRC_TAPEU      Unit type for tape
    248     (F8) CHARACTER       24   APQRC_TIME
    248     (F8) CHARACTER        2    APQRC_TZDEF     Input offset default
    250     (FA) CHARACTER        5    APQRC_TMFMT     Time format options
    255     (FF) CHARACTER        1    *               Reserved
    256    (100) SIGNED           2    APQRC_TPREC     Time-stamp precision
    258    (102) CHARACTER       12    APQRC_LOGRT     Minimum log retention period
    270    (10E) SIGNED           2    APQRC_TZNUM     Number of entries in time zone label table
    272    (110) CHARACTER        8   APQRC_TZTBL(32)  Time zone label table
    528    (210) BIT(16)          2   APQRC_TROPT      DBRC trace options
    530    (212) CHARACTER        5   APQRC_IMSPLEX    IMSplex name
    535    (217) CHARACTER        5   *                Reserved
    540    (21C) UNSIGNED         4   APQRC_SIZW_DSNUM   SIZEALERT dsnum
    544    (220) UNSIGNED         4   APQRC_SIZW_VOLNUM  SIZEALERT volnum
    548    (224) UNSIGNED         4   APQRC_SIZW_PERCENT SIZEALERT percent
    552    (228) UNSIGNED         4   APQRC_LOGW_DSNUM   LOGALERT dsnum
    556    (22C) UNSIGNED         4   APQRC_LOGW_VOLNUM  LOGALERT volnum
    560    (230) UNSIGNED        44   APQRC_CMDRNQ       CMDAUTH RECON qual
    604    (25C) UNSIGNED         8   APQRC_DBCOUNT      REGISTERED DBCOUNT
   612    (264) CHARACTER        8   APQRC_CATLG        CATALOG NAME (VERSION=6.0)
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE         LENGTH  NAME (DIM)                DESCRIPTION
======== ======== ========= ========  ==============            ================================
       0      (0) STRUCTURE        53  APQRC_RECONDS_INFO
       0      (0) CHARACTER         8   APQRC_RECONDS_DDNAME     RECON DD name
       8      (8) CHARACTER        44   APQRC_RECONDS_DSNAME     RECON DS name
      52     (34) BIT(8)            1   APQRC_RECONDS_STATUS     RECON DS status
                 1... ....              APQRC_RECONDS_COPY1     COPY 1
                 .1.. ....              APQRC_RECONDS_COPY2     COPY 2
                 ..1. ....              APQRC_RECONDS_SPARE     Spare
                 ...1 ....              APQRC_RECONDS_DISCARDED Discarded
                 .... 1...              APQRC_RECONDS_UNAVAIL   Unavailable
CONSTANTS

  LEN  TYPE           VALUE           NAME                      DESCRIPTION
=====  =========  ================  ==============            ================================
    8  CHARACTER   DSPAPQRC          APQRC_EYECATCHER
```

Related concepts

“Macro forms of the DSPAPI macro” on page 344

There are four different macro forms for the DSPAPI macro: Standard (S), List (L), Modify (M), and Execute (E), with two variations, COMPLETE and NOCHECK. The List, Modify, and Execute forms are usually used in combinations when writing reentrant programs or when the application issues multiple requests.

Related reference

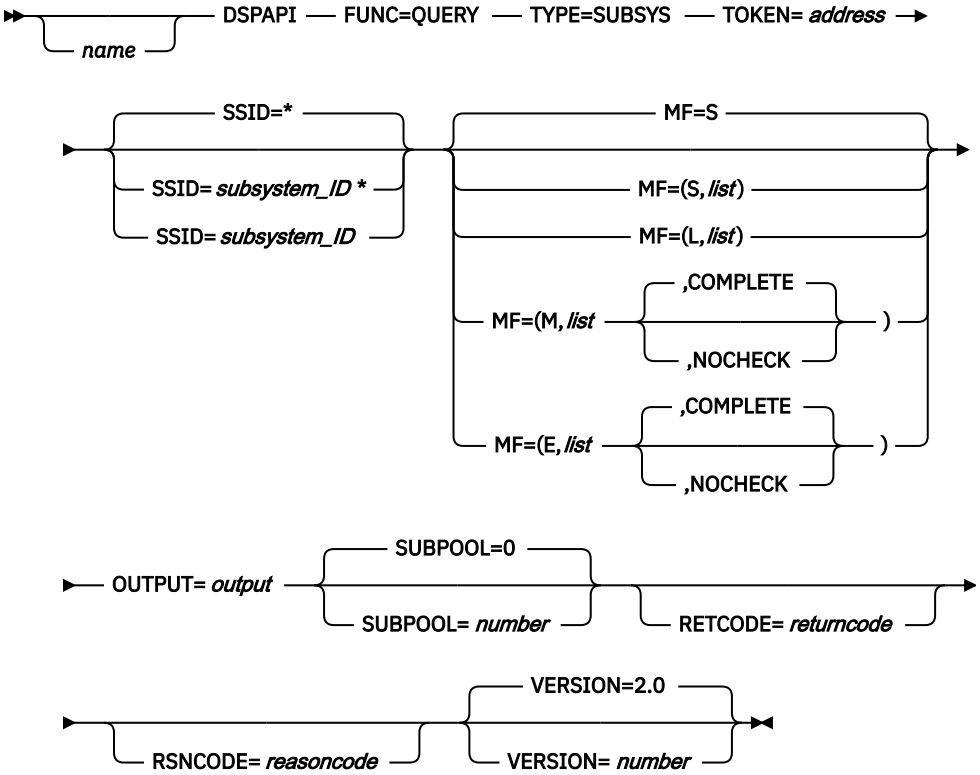
“DBRC query request (QUERY)” on page 367

You can use the DBRC Query request (DSPAPI FUNC=QUERY) along with the TYPE parameter to retrieve various types of information from the RECON data set.

Subsystem query request (TYPE=SUBSYS)

You can use the Subsystem query (DSPAPI FUNC=QUERY TYPE=SUBSYS) request to retrieve subsystem information from the RECON data set for a specific subsystem or all subsystems.

Syntax for the TYPE=SUBSYS query request



Parameters for the TYPE=SUBSYS query request

name
Optional symbol you can specify. If used, begins in column 1.

TYPE=SUBSYS
Specifies that subsystem information is requested.

TOKEN=*address* | (2 - 12)
Specifies the address of a 4-byte field that was returned on the FUNC=STARTDBRC request.

SSID= * | *name | *name* | (2 - 12)**
Specifies the name of the subsystem being queried. You can use the wildcard keyword * (an asterisk) alone to request information about all subsystems (SSID=*, the default). You can also use it at the end of a name to query subsystems whose names match the pattern. In this case, the asterisk must be preceded by at least one alphabetic character.

SSYTYPE=ALL | ONLINE | BATCH | DBRCAPI
Specifies the type of subsystem for which information is being requested. You cannot specify this parameter if you also specify a specific subsystem name for the SSID parameter or if you specify TRACKER=YES. SSTYPE=DBRCAPI requires that you specify a minimum version number of VERSION=2.0. SSTYPE=ALL is the default.

MF=S | L | M | E
Optional parameter that specifies the macro form of the request.

OUTPUT=*output* | (2 - 12)
Specifies a field to receive a pointer to the first block of a possible chain of subsystem information blocks.

The output address is 0 if no output was built which can occur if nothing in the RECON satisfies the request or if an error occurs before any output is built.

The storage for the output blocks is not preallocated by the caller. DBRC acquires storage from the specified subpool for these blocks. The caller must free this storage using the Buffer Release service (DSPAPI FUNC=RELBUF) and specify the returned output address.

SUBPOOL= 0 | *number*
Specifies the subpool number for the storage being obtained. If not specified, the default is the subpool specified by the FUNC=STARTDBRC request. Otherwise, subpool 0 is the default.

RETCODE=*returncode* | (2 - 12)
Specifies a place in storage to receive the return code. If specified as a symbol, the symbol must be the label of a word of storage. If specified as a register, the register must contain the address of a word of storage. If not specified, the return code is placed in register 15.

RSNCODE=*reasoncode* | (2 - 12)
Specifies a place in storage to receive the reason code. If specified as a symbol, the symbol must be the label of a word of storage. If specified as a register, the register must contain the address of a word of storage. If not specified, the reason code is placed in register 0.

VERSION=2.0 | *number*
Specifies the version number of the parameter list to be generated by this request.

To use the parameters associated with a version, you must specify the number of that version or a later version. If you specify an earlier version level, the parameter is not accepted by macro processing, and an error message is issued at assembly time. If parameters have a version dependency, the parameter descriptions with each request type identify the version number required.

Valid version numbers for the FUNC=QUERY TYPE=SUBSYS request are 1.0 and 2.0.

Return and reason codes for the TYPE=SUBSYS query request

Table 95. Return and reason codes for TYPE=SUBSYS query requests

Code type	Return codes	Reason codes	Meaning
	X'00000000'	X'00000000'	Request completed successfully.
Warning	X'00000008'	X'D8600001'	No subsystem record of the requested type, active or tracker, exists.
	X'00000008'	X'D8600002'	No batch or online subsystem records exist.
Severe error	X'0000000C'	X'C9000001'	Invalid TOKEN. The TOKEN block passed to the API is not recognized as a TOKEN created by a FUNC=STARTDBRC call.
	X'0000000C'	X'C900000A'	The TCB address is not the same as the TCB address under which the STARTDBRC service was issued.
	X'0000000C'	X'D8600100'	Security error. SAF or the DBRC Command Authorization exit routine (DSPDCAX0) determined that the user is not authorized to perform the request.
Storage error	X'00000028'	X'D8600001'	Error obtaining storage for SUBSYS block.
	X'00000028'	X'D9100001'	An error occurred processing the request. DBRC will release storage that was obtained up to this point. However, another error was encountered during the attempt to release storage.

Table 95. Return and reason codes for TYPE=SUBSYS query requests (continued)

Code type	Return codes	Reason codes	Meaning
Internal error	X'0000002C'	X'D8000001'	RECON open failure.
	X'0000002C'	X'D8600001'	Failure attempting to locate the first or the specified subsystem record of the requested type, active or tracker.
	X'0000002C'	X'D8600002'	Failure attempting to locate the next subsystem record of the requested type, active or tracker.
Parameter error	X'00000030'	X'C9000001'	The function (FUNC) specified in the parameter list passed to the API is invalid.
	X'00000030'	X'C9000002'	Invalid TOKEN field address. The address of the field containing the API TOKEN failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000003'	Invalid RETCODE field address. The address of the field to contain the API RETCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000004'	Invalid RSNCODE field address. The address of the field to contain the API RSNCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000005'	Invalid OUTPUT field address. The address of the field to contain the OUTPUT address failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000010'	Invalid SSID field address. The address of the field containing the SSID failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'D8000001'	Missing or invalid OUTPUT parameter.
	X'00000030'	X'D8000002'	Invalid value specified for TYPE parameter.
	X'00000030'	X'D8600001'	SSTYPE=ONLINE|BATCH|DBRCAPI cannot be specified on requests for a specific SSID or for a tracking subsystem request (TRACKER=YES).
	X'00000030'	X'D8600100'	When using a wildcard, at least one alphabetic character must precede the asterisk.
	X'00000030'	X'D8600101'	When using a wildcard, the asterisk must be the last character.

Output for the TYPE=SUBSYS query request

The following figure illustrates the format of the output from a QUERY TYPE=SUBSYS request. The following DSECT sample describes in detail the fields of the storage blocks and their relationship to each other.

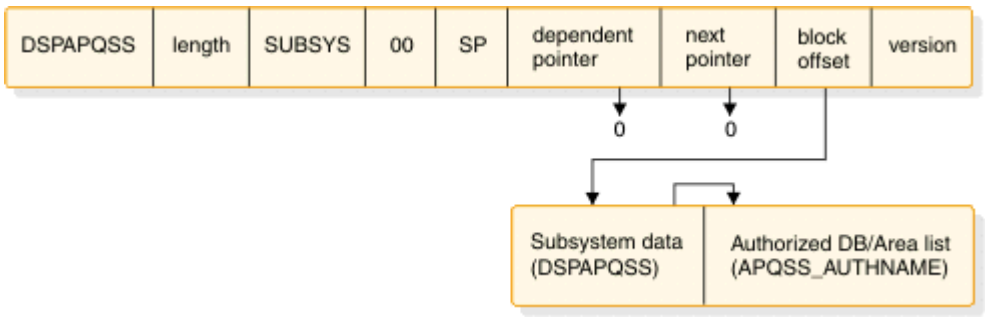


Figure 27. Format for QUERY TYPE=SUBSYS output

DSECT of DSPAPQSS

OFFSET DECIMAL	OFFSET HEX	TYPE	LENGTH	NAME (DIM)	DESCRIPTION
0	(0)	STRUCTURE	64	DSPAPQSS	
0	(0)	CHARACTER	8	APQSS_SSID	Subsystem identifier
8	(8)	UNSIGNED	4	APQSS_AUTHLIST	Offset to authd DB/Area list
12	(C)	SIGNED	4	APQSS_AUTHCOUNT	Number of authorized DB/Areas
16	(10)	UNSIGNED	2	APQSS_AUTHLEN	Length of auth entry
18	(12)	CHARACTER	6	*	Reserved
24	(18)	CHARACTER	12	APQSS_LOGTIME	Start time of log
36	(24)	UNSIGNED	1	APQSS_RELLVL	Subsystem release level '71'X=V7R1,'81'X=V8R1 '91'X=V9R1, etc.
37	(25)	CHARACTER	1	APQSS_COEXLVL	Coexistence level
38	(26)	UNSIGNED	1	APQSS_IRLMCT	IRLM status count
39	(27)	CHARACTER	1	*	Reserved
40	(28)	CHARACTER	8	APQSS_GSGNAME	Global Service Group name
48	(30)	CHARACTER	5	APQSS_IRLMID	IRLM ID of SS
53	(35)	CHARACTER	5	APQSS_IRLMBK	IRLM ID of backup SS
58	(3A)	BIT(8)	1	APQSS_FLAGS	Flags
		1...		APQSS_TYPE	1=Online | 0=batch
		.1..		APQSS_ABTERM	Abnormal termination
		..1.		APQSS_RCVPRC	Recovery processing started
		...1		APQSS_BKSIGN	Backup SS signed on
		.... 1...		APQSS_IRLMFL	IRLM failure
		.... .1..		APQSS_COMMFL	COMM failure
		.... ..1.		APQSS_SYSFL	SYS failure
		1		APQSS_ACTVST	Status of active SS when backup exists, 1=abterm
59	(3B)	BIT(8)	1	APQSS_FLAGS2	FLAGS 2
		1...		APQSS_SHRING	Sharing covered DBs
		.1..		APQSS_TRKER	Subsystem is a Tracker
		..1.		APQSS_TRKTRM	TRACKER has terminated
		...1		APQSS_TRCKED	SSID is tracked
		.... 1...		APQSS_FRSTSO	1ST signon after RSR takeover is in progress
		.... .1..		APQSS_XRFCAP	SS is XRF capable
		.... ..1.		APQSS_DBRCAPI	SS is a DBRC application (VERSION=2.0)
		1		APQSS_BPE	BPE-based subsystem (VERSION=4.0)
		11		*	Reserved
60	(3C)	SIGNED	2	APQSS_BCKTKN	Backup recovery token
62	(3E)	CHARACTER	2	*	Reserved

OFFSET DECIMAL	OFFSET HEX	TYPE	LENGTH	NAME (DIM)	DESCRIPTION
0	(0)	STRUCTURE	32	APQSS_AUTHNAME	Names of authd DB/Areas
0	(0)	CHARACTER	8	APQSS_DBNAME	DB name
8	(8)	CHARACTER	8	APQSS_AREANM	If FP, Area name
16	(10)	UNSIGNED	1	APQSS_SHRLVL	Share level
17	(11)	UNSIGNED	1	APQSS_DBACCS	Access intent
18	(12)	UNSIGNED	1	APQSS_DBNCOD	Encoded state
19	(13)	UNSIGNED	1	APQSS_DBSTAT	DB status flags
20	(14)	UNSIGNED	2	APQSS_DBEQCT	DB EQE count
22	(16)	SIGNED	2	APQSS_GLBDMB	Global DMB number
24	(18)	BIT(8)	1	APQSS_AUTHFLAGS	Flags
		1...		APQSS_NRDBUP	Nonrecov DB/Area updated
		.1..		APQSS_COVRD	DB covered by GSG
		..1.		APQSS_NRECV	nonrecoverable DB/Area
		...1		APQSS_ORDBDS	0=A-J/1=M-V ACTIVE

```
                  .... 1...              APQSS_OLRON     0 = no OLR active
                  .... .1..              APQSS_OLROWR    0 = no OLR owner
                  .... ..1.              APQSS_OLROWD    0 OLR not owned by SS
      25     (19) CHARACTER        7   *                 Reserved

CONSTANTS

  LEN  TYPE           VALUE           NAME               DESCRIPTION
=====  =========  ================  ===============    ================================
    8  CHARACTER   DSPAPQSS          APQSS_EYECATCHER
```

Related concepts

“Macro forms of the DSPAPI macro” on page 344
There are four different macro forms for the DSPAPI macro: Standard (S), List (L), Modify (M), and Execute (E), with two variations, COMPLETE and NOCHECK. The List, Modify, and Execute forms are usually used in combinations when writing reentrant programs or when the application issues multiple requests.

Related reference

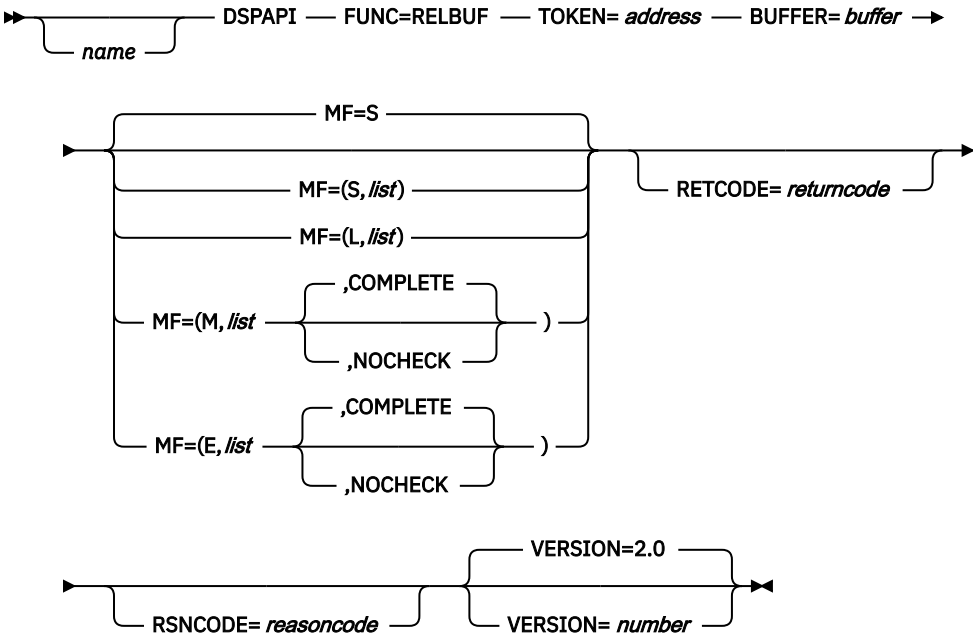
“DBRC query request (QUERY)” on page 367
You can use the DBRC Query request (DSPAPI FUNC=QUERY) along with the TYPE parameter to retrieve various types of information from the RECON data set.

Chapter 33. DBRC release buffer request (RELBUF)

You can use the DBRC release buffer request (DSPAPI FUNC=RELBUF) to release storage that was acquired as a result of DBRC query, command, authorization, and unauthorization requests.

Each request returns a chain of one or more blocks containing the requested information. It is the caller's responsibility to ensure that the storage DBRC allocated for these blocks is freed.

Syntax for the RELBUF request



Parameters for the RELBUF request

name
Optional symbol you can specify. If used, begins in column 1.

TOKEN=*address* | (2 - 12)
Required parameter that specifies the address of a 4-byte field that was returned on the FUNC=STARTDBRC request.

BUFFER=*buffer* | (2 - 12)
Specifies a field containing the address of the first block of a chain of one or more information blocks to release. This storage was acquired by DBRC in order to satisfy another DBRC request.

RETCODE=*returncode* | (2 - 12)
Specifies a place in storage to receive the return code. If specified as a symbol, the symbol must be the label of a word of storage. If specified as a register, the register must contain the address of a word of storage. If not specified, the return code is placed in register 15.

RSNCODE=*reasoncode* | (2 - 12)
Specifies a place in storage to receive the reason code. If specified as a symbol, the symbol must be the label of a word of storage. If specified as a register, the register must contain the address of a word of storage. If not specified, the reason code is placed in register 0.

MF=S | L | M | E
Specifies the macro form of the request.

VERSION=2.0 | *number*
Specifies the version number of the parameter list to be generated by this request.

To use the parameters associated with a version, you must specify the number of that version or a later version. If you specify an earlier version level, the parameter is not accepted for processing and an error message is issued at assembly time. If parameters have a version dependency, the parameter descriptions with each request type identify the version number required.

Return and reason codes for RELBUF

Table 96. Return and reason codes for RELBUF

Code type	Return codes	Reason codes	Meaning
	X'00000000'	X'00000000'	Request completed successfully.
Partial success	X'00000004'	X'D9000001'	There was no storage to free. An address of zero was passed in the BUFFER parameter.
Severe error	X'0000000C'	X'C9000001'	Invalid TOKEN. The TOKEN block passed to the API is not recognized as a TOKEN created by a FUNC=STARTDBRC call.
	X'0000000C'	X'C900000A'	The TCB address is not the same as the TCB address under which the STARTDBRC request was issued.
Storage error	X'00000028'	X'D9100001'	An error occurred attempting to release storage.
Parameter error	X'00000030'	X'C9000001'	Parameter error. The function (FUNC) specified in the parameter list passed to the API is invalid.
	X'00000030'	X'C9000002'	Invalid TOKEN field address. The address of the field containing the API TOKEN failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000003'	Invalid RETCODE field address. The address of the field to contain the API RETCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000004'	Invalid RSNCODE field address. The address of the field to contain the API RSNCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000007'	Invalid BUFFER address. The address of the BUFFER to be released failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'D9100001'	The header of a storage block to be freed is invalid.
	X'00000030'	X'D9100002'	The length of a storage block to be freed is not greater than zero.

Related concepts

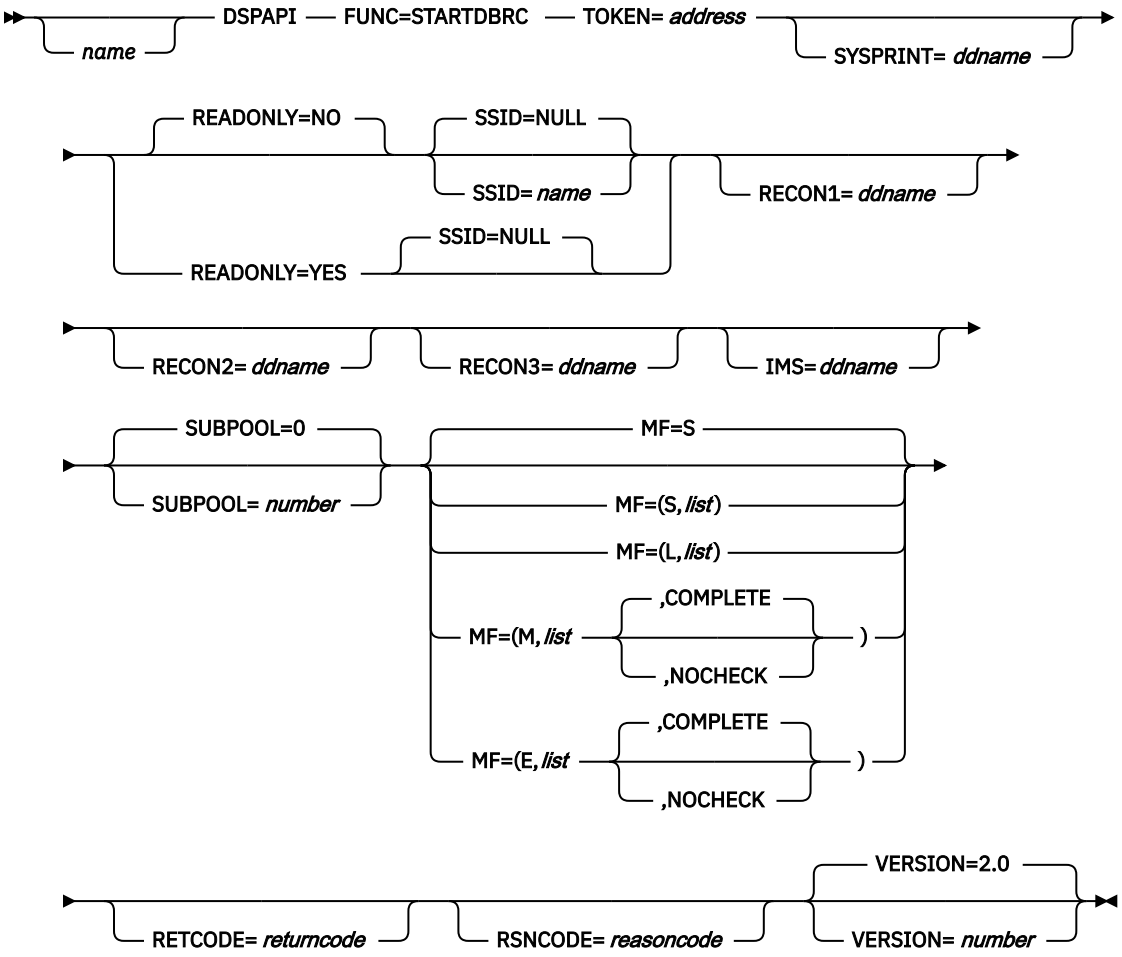
“DBRC API” on page 341
Your applications can obtain services from Database Recovery Control (DBRC) through the DBRC application programming interface (API), a release-independent, assembler macro interface. The application obtains these services by issuing DBRC API requests to DBRC, and DBRC returns the results to an area in storage where the application can retrieve them.

“Macro forms of the DSPAPI macro” on page 344
There are four different macro forms for the DSPAPI macro: Standard (S), List (L), Modify (M), and Execute (E), with two variations, COMPLETE and NOCHECK. The List, Modify, and Execute forms are usually used in combinations when writing reentrant programs or when the application issues multiple requests.

Chapter 34. DBRC start request (STARTDBRC)

You can use the DBRC start request (STARTDBRC) to initialize the DBRC API and to start DBRC.

Syntax for the STARTDBRC request



Parameters for STARTDBRC

name
Optional symbol you can specify. If used, begins in column 1.

TOKEN=*address* | (2 - 12)
Specifies the address of a 4-byte field to receive the API token. This token must be included in all subsequent requests associated with this STARTDBRC request.

SYSPRINT= *ddname* | (2 - 12)
Specifies the address of an 8-byte field that contains the DD name of an output data set to be used for messages. If omitted, the default name SYSPRINT is used.

READONLY=NO | YES
Specifies whether (YES) or not (NO) the application needs read only access to the DBRC information. READONLY=NO requires that the application has a minimum of UPDATE level authority to the RECON data sets.

To use this parameter, you must specify API VERSION=2.0 or later.

SSID=NULL | *name*
Specifies the address of an 8-byte field that contains the subsystem name that is used to register with DBRC. If you specify SSID=NULL, registration with DBRC is not done. Do not specify both READONLY=YES and SSID=*symbol*.

The default is SSID=NULL. To use this parameter, you must specify API VERSION=2.0 or later.

RECON1=*ddname* | (2 - 12)
Specifies the address of an 8-byte field that contains the DD name that is used in place of the default DD name RECON1.

If omitted, the default name RECON1 is used. To use this parameter, you must specify API VERSION=2.0 or later.

RECON2=*ddname* | (2 - 12)
Specifies the address of an 8-byte field that contains the DD name that is used in place of the default DD name RECON2.

If omitted, the default name RECON2 is used. To use this parameter, you must specify API VERSION=2.0 or later.

RECON3=*ddname* | (2 - 12)
Specifies the address of an 8-byte field that contains the DD name that is used in place of the default DD name RECON3.

If omitted, the default name RECON3 is used. To use this parameter, you must specify API VERSION=2.0 or later.

IMS=IMS | *ddname*
Specifies the address of an 8-byte field that contains the DD name that is used in place of the default DD name IMS.

The default is IMS=IMS. To use this parameter, you must also specify API VERSION=2.0 or later.

SUBPOOL=0 | *number*
Specifies the default subpool number that is to be used for all requests (for example QUERY and AUTH) that return storage. For information on valid subpools for your program, see the *z/OS MVS Assembler Services Guide*.

SUBPOOL=0 is the default for the request if SUBPOOL is not specified here or on the request.

MF=S | L | M | E
Specifies the macro form of the request.

RETCODE=*returncode* | (2 - 12)
Specifies a place in storage to receive the return code. If specified as a symbol, the symbol must be the label of a word of storage. If specified as a register, the register must contain the address of a word of storage. If not specified, the return code is placed in register 15.

RSNCODE=*reasoncode* | (2 - 12)
Specifies a place in storage to receive the reason code. If specified as a symbol, the symbol must be the label of a word of storage. If specified as a register, the register must contain the address of a word of storage. If not specified, the reason code is placed in register 0.

VERSION=2.0 | *number*
Specifies the version number of the parameter list that is generated by this request.

To use the parameters associated with a version, you must specify the number of that version or a later version. If you specify an earlier version level, the parameter is not accepted for processing and an error message is issued at assembly time. If parameters have a version dependency, the parameter descriptions with each request type identify the required version number.

Return and reason codes for STARTDBRC

Table 97. Return and reason codes for the STARTDBRC request

Code type	Return codes	Reason codes	Meaning
	X'00000000'	X'00000000'	Request completed successfully.
Warning	X'00000004'	X'E2000001'	The RECON data sets are not initialized. The only function allowed is an INIT.RECON command.
	X'00000004'	X'E2000002'	The RECON data sets are not upgraded. The only function allowed is a CHANGE.RECON UPGRADE command.
Severe error	X'0000000C'	X'C9000002'	DBRC could not successfully initialize during phase 0 of DBRC initialization.
	X'0000000C'	X'C9000003'	DBRC could not successfully initialize during phase 1 of DBRC initialization.
	X'0000000C'	X'C9000004'	A z/OS LOAD failed for the DBRC module DSPCRTR0.
	X'0000000C'	X'C9000005'	STORAGE request failure. The API cannot obtain storage necessary to complete the request.
	X'0000000C'	X'C9000012'	STARTDBRC has been previously issued without an intervening STOPDBRC.
	X'0000000C'	X'C9D40001'	DSPAPI macro load failed for DBRC module DSPAPI00.
	X'0000000C'	X'E2000001'	The RECON data sets are not initialized. STARTDBRC fails because an SSID was supplied and SIGNON cannot be performed.
	X'0000000C'	X'E2000002'	The RECON data sets are not upgraded. STARTDBRC fails because an SSID was supplied and SIGNON cannot be performed.
	X'0000000C'	X'E2000003'	STARTDBRC specified READONLY. However, an SSID was supplied and SIGNON cannot be performed.
	X'0000000C'	X'E200000A'	RECON is either not initialized or not upgraded and V1 is the caller.
	X'0000000C'	X'E2000100'	Security error. SAF or the DBRC Command Authorization exit routine (DSPDCAX0) determined that the user is not authorized to perform the request.
	X'0000000C'	X'E210nnnn'	Error attempting to sign on the application. 'nnnn' is the return code from DBRC Signon module, DSPSSIGN.
Internal error	X'00000028'	X'C9000001'	DBRC internal error. Failure writing the subsystem record.
Parameter error	X'00000030'	X'C9000001'	Parameter error. The function (FUNC) specified in the parameter list passed to the API is invalid.
	X'00000030'	X'C9000002'	Invalid TOKEN field address. The address of the field to contain the API TOKEN failed validity checking. The address specifies storage not owned by the calling program.

Table 97. Return and reason codes for the STARTDBRC request (continued)

Code type	Return codes	Reason codes	Meaning
	X'00000030'	X'C9000003'	Invalid RETCODE field address. The address of the field to contain the API RETCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000004'	Invalid RSNCODE field address. The address of the field to contain the API RSNCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000006'	Invalid SYSPRINT field address. The address of the field containing the SYSPRINT DD name failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C900000A'	Invalid IMS field address. The address of the field that contains the IMS DD name failed validity checking.
	X'00000030'	X'C900000C'	Invalid RECON1 field address. The address of the field that contains the RECON1 DD name failed validity checking. The address specifies storage that is not owned by the calling program.
	X'00000030'	X'C900000D'	Invalid RECON2 field address. The address of the field that contains the RECON2 DD name failed validity checking. The address specifies storage that is not owned by the calling program.
	X'00000030'	X'C900000E'	Invalid RECON3 field address. The address of the field that contains the RECON3 DD name failed validity checking. The address specifies storage that is not owned by the calling program.
	X'00000030'	X'E2100001'	An SSID of zeroes or blanks was specified.

Related concepts

"DBRC API" on page 341

Your applications can obtain services from Database Recovery Control (DBRC) through the DBRC application programming interface (API), a release-independent, assembler macro interface. The application obtains these services by issuing DBRC API requests to DBRC, and DBRC returns the results to an area in storage where the application can retrieve them.

"Macro forms of the DSPAPI macro" on page 344

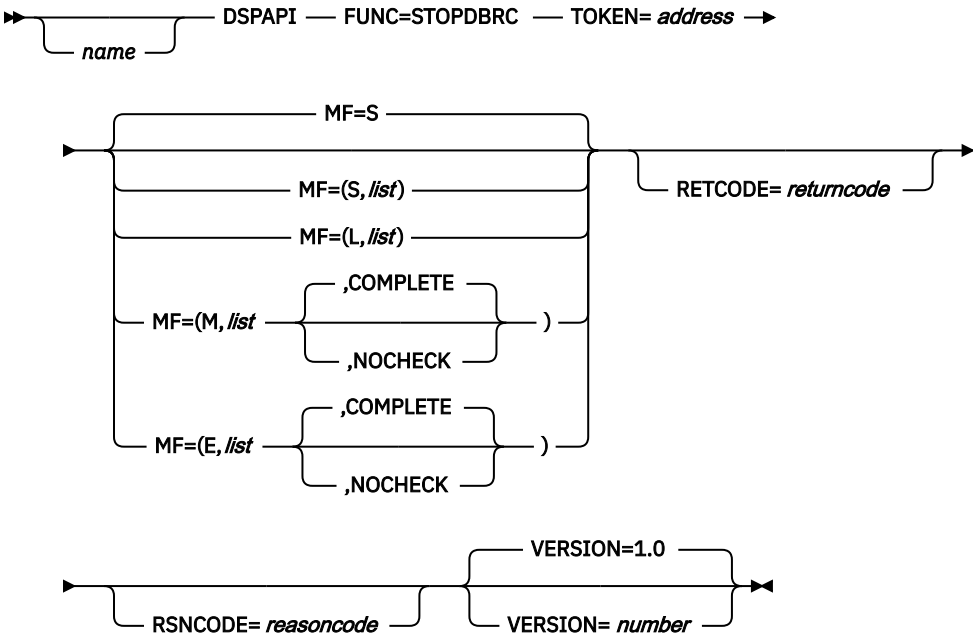
There are four different macro forms for the DSPAPI macro: Standard (S), List (L), Modify (M), and Execute (E), with two variations, COMPLETE and NOCHECK. The List, Modify, and Execute forms are usually used in combinations when writing reentrant programs or when the application issues multiple requests.

Chapter 35. DBRC stop request (STOPDBRC)

You can use the STOPDBRC request to terminate the DBRC application and to stop DBRC.

If the application registered to DBRC by supplying an SSID on the STARTDBRC request, a subsystem record for the application was recorded in RECON. The STOPDBRC request automatically deregisters the application, unauthorizing any authorized databases and deleting the subsystem record.

Syntax for the STOPDBRC request



Parameters for STOPDBRC

name
Optional symbol you can specify. If used, begins in column 1.

TOKEN=*symbol* | (2 - 12)
Specifies the address of a 4-byte field that was returned on the FUNC=STARTDBRC request.

RETCODE=*returncode* | (2-12)
If specified as a symbol, specifies the label of a word of storage to receive the return code. If specified as a register, the register must contain the address of a word of storage to receive the return code. Regardless of whether RETCODE is specified, register 15 contains the return code.

RSNCODE=*reasoncode* | (2-12)
If specified as a symbol, the symbol must be the label of a word of storage to receive the reason code. If specified as a register, the register must contain the address of a word of storage to receive the reason code. Regardless of whether RSNCODE is specified, register 0 contains the reason code.

MF=S | L | M | E
Specifies the macro form of the request.

VERSION=1.0 | *number*
Specifies the version number of the parameter list to be generated by this request.

To use the parameters associated with a version, you must specify the number of that version or a later version. If you specify an earlier version level, the parameter is not accepted for processing and an error message is issued at assembly time. If parameters have a version dependency, the parameter descriptions with each request type identify the version number required.

Return and reason codes for STOPDBRC

Table 98. Return and reason codes for the STOPDBRC request

Code type	Return codes	Reason codes	Meaning
	X'00000000'	X'00000000'	Request completed successfully.
Severe error	X'0000000C'	X'C9000001'	Invalid TOKEN. The TOKEN block passed to the API is not recognized as a TOKEN created by a FUNC=STARTDBRC call.
	X'0000000C'	X'C900000A'	The TCB address is not the same as the TCB address under which the STARTDBRC request was issued.
	X'0000000C'	X'E220nnnn'	Error attempting to sign off the application. 'nnnn' is the return code from DBRC Signoff module, DSPSSIGN.
	X'0000000C'	X'E2000100'	Security error. SAF or the DBRC Command Authorization exit routine (DSPDCAX0) determined that the user is not authorized to perform the request.
Parameter error	X'00000030'	X'C9000001'	Parameter error. The function (FUNC) specified in the parameter list passed to the API is invalid.
	X'00000030'	X'C9000002'	Invalid TOKEN field address. The address of the field containing the API TOKEN failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000003'	Invalid RETCODE field address. The address of the field to contain the API RETCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000004'	Invalid RSNCODE field address. The address of the field to contain the API RSNCODE failed validity checking. The address specifies storage not owned by the calling program.

Related concepts

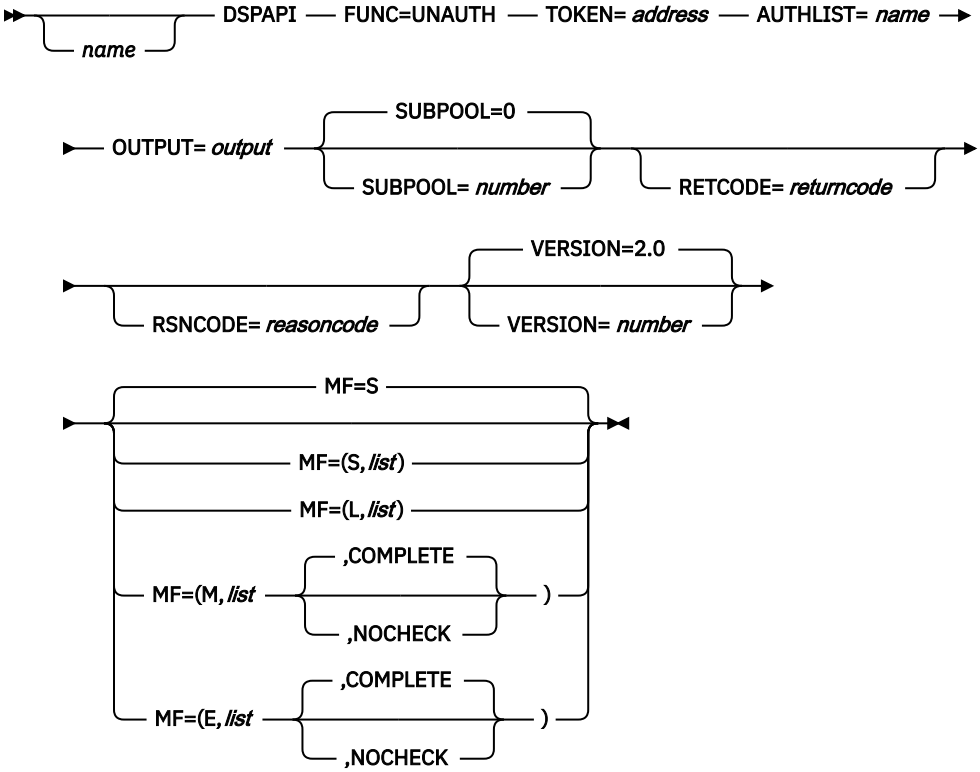
"DBRC API" on page 341
Your applications can obtain services from Database Recovery Control (DBRC) through the DBRC application programming interface (API), a release-independent, assembler macro interface. The application obtains these services by issuing DBRC API requests to DBRC, and DBRC returns the results to an area in storage where the application can retrieve them.

"Macro forms of the DSPAPI macro" on page 344
There are four different macro forms for the DSPAPI macro: Standard (S), List (L), Modify (M), and Execute (E), with two variations, COMPLETE and NOCHECK. The List, Modify, and Execute forms are usually used in combinations when writing reentrant programs or when the application issues multiple requests.

Chapter 36. DBRC unauthorization request (UNAUTH)

You can use the UNAUTH request to explicitly remove authorization to a database or area. Authorization by an application is implicitly removed by the STOPDBRC request. UNAUTH is the opposite of FUNC=AUTH.

Syntax for the UNAUTH request



Parameters for the UNAUTH request

name *name*
Optional parameter. Begin name in column 1.

TOKEN=*address* | (2-12)
Specifies the address of the API token which was returned on the FUNC=STARTDBRC macro.

AUTHLIST=*name* | (2 - 12)
Specifies the list of database names or Fast Path areas to be unauthorized. The list consists of a fullword containing the number of elements in the list, a fullword containing the length of an element, followed by one or more elements. Each element consists of an 8 character DB name or Fast Path DEDB name and 8 characters of blanks (X'40') or a Fast Path area name.

OUTPUT=*output* | (2-12)
Specifies a field to receive a pointer to the authorization output block DSPAPAUB.

The output address is zero if no output was built. This can happen if an error occurs before any output could be built.

The storage for the output blocks is not pre-allocated by the caller. DBRC acquires storage from the specified subpool for these blocks. It is the responsibility of the caller to free this storage using the Buffer Release service (DSPAPI FUNC=RELBUF) specifying the returned output address.

SUBPOOL= 0 | *number*
Specifies the subpool number for the storage being obtained. See the z/OS MVS Programming: Assembler Services Guide for information on valid subpools for your program. If not specified, the default is the subpool specified by the FUNC=STARTDBRC request. Otherwise, subpool 0 is the default.

RETCODE=*returncode* | (2-12)
If specified as a symbol, specifies the label of a word of storage to receive the return code. If specified as a register, the register must contain the address of a word of storage to receive the return code. Regardless of whether RETCODE is specified, register 15 contains the return code.

RSNCODE=*reasoncode* | (2-12)
If specified as a symbol, the symbol must be the label of a word of storage to receive the reason code. If specified as a register, the register must contain the address of a word of storage to receive the reason code. Regardless of whether RSNCODE is specified, register 0 contains the reason code.

VERSION=2.0 | *number*
Specifies the version number of the parameter list to be generated by this macro.

To use the parameters associated with a version, you must specify the number of that version or a later version. If you specify an earlier version level, the parameter is not accepted by macro processing, and an error message is issued at assembly time. If parameters have a version dependency, the parameter descriptions with each request type identify the version number required.

The valid version number for the FUNC=UNAUTH request is 2.0 (the default).

MF=S | L | M | E
Specifies the macro form of the request.

Return and reason codes for UNAUTH

Table 99. DSPAPI FUNC=UNAUTH request return and reason codes

Code type	Return code	Reason code	Meaning
	X'00000000'	X'00000000'	Request completed successfully.
Warning	X'00000008'	X'C1000001'	One or more entries in the AUTHLIST could not be processed. A reason code has be set in the corresponding entry in the UNAUTH output block.
Severe error	X'0000000C'	X'C1000001'	Application is not signed on to DBRC.
	X'0000000C'	X'C1000004'	UNAUTH processing could not complete because the application is not signed; no SS rcd was found. This should not occur under normal conditions since an earlier check of the GDB indicated the SS was signed on.
	X'0000000C'	X'C9000001'	Invalid TOKEN. The TOKEN block passed to the API is not recognized as a TOKEN created by a FUNC=STARTDBRC call.
	X'0000000C'	X'C900000A'	The TCB address is not the same as the TCB address under which the STARTDBRC service was issued.
Storage error	X'00000028'	X'C1000001'	Error obtaining storage for the UNAUTH output block.
Internal error	X'0000002C'	X'C1000001'	Error attempting to start RECON multiple update processing.
	X'0000002C'	X'C1000002'	Error attempting to end RECON multiple update processing.

Table 99. DSPAPI FUNC=UNAUTH request return and reason codes (continued)

Code type	Return code	Reason code	Meaning
	X'0000002C'	X'C1000006'	Entry in UNAUTH output block could not be found. This should not occur.
	X'0000002C'	X'C1000007'	Internal error encountered during DBRC unauthorization processing.
	X'0000002C'	X'C1000008'	Internal error encountered during DBRC unauthorization processing–invalid parameters.
Parameter error	X'00000030'	X'C1000001'	No AUTHLIST passed.
	X'00000030'	X'C1000002'	AUTHLIST passed with no entries.
	X'00000030'	X'C1000003'	Duplicate elements in AUTHLIST.
	X'00000030'	X'C1000004'	Missing or invalid OUTPUT parameter.
	X'00000030'	X'C9000001'	The function (FUNC) specified in the parameter list passed to the API is invalid.
	X'00000030'	X'C9000002'	Invalid TOKEN address. The address of the field containing the API TOKEN failed validity checking. The address specifies storage not owned by the calling program
	X'00000030'	X'C9000003'	Invalid RETCODE address. The address of the field containing the API RETCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000004'	Invalid RSNCODE address. The address of the field containing the API RSNCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000005'	Invalid OUTPUT address. The address of the field containing the API OUTPUT failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C900000A'	An incorrect VERSION value was specified for the requested function (FUNC).
	X'00000030'	X'C900001A'	Invalid AUTHLIST address. The address of the field containing the API AUTHLIST failed validity checking. The address specifies storage not owned by the calling program.

Related concepts

"DBRC API" on page 341

Your applications can obtain services from Database Recovery Control (DBRC) through the DBRC application programming interface (API), a release-independent, assembler macro interface. The application obtains these services by issuing DBRC API requests to DBRC, and DBRC returns the results to an area in storage where the application can retrieve them.

"Macro forms of the DSPAPI macro" on page 344

There are four different macro forms for the DSPAPI macro: Standard (S), List (L), Modify (M), and Execute (E), with two variations, COMPLETE and NOCHECK. The List, Modify, and Execute forms are usually used in combinations when writing reentrant programs or when the application issues multiple requests.

Related reference

"APAUB_RsnCode for UNAUTH output block" on page 449
You can use this table to search for APAUB_RsnCode values for UNAUTH request return and reason codes. Each code is accompanied by an explanation of the code.

DBRC request return codes (Messages and Codes)

Return and reason codes for UNAUTH

You can use this table to search for reason and return codes for the DBRC UNAUTH request. Each code is accompanied by the code type and an explanation of the code.

Table 100. DSPAPI FUNC=UNAUTH request return and reason codes

Code type	Return code	Reason code	Meaning
	X'00000000'	X'00000000'	Request completed successfully.
Warning	X'00000008'	X'C1000001'	One or more entries in the AUTHLIST could not be processed. A reason code has be set in the corresponding entry in the UNAUTH output block.
Severe error	X'0000000C'	X'C1000001'	Application is not signed on to DBRC.
	X'0000000C'	X'C1000004'	UNAUTH processing could not complete because the application is not signed; no SS rcd was found. This should not occur under normal conditions since an earlier check of the GDB indicated the SS was signed on.
	X'0000000C'	X'C9000001'	Invalid TOKEN. The TOKEN block passed to the API is not recognized as a TOKEN created by a FUNC=STARTDBRC call.
	X'0000000C'	X'C900000A'	The TCB address is not the same as the TCB address under which the STARTDBRC service was issued.
Storage error	X'00000028'	X'C1000001'	Error obtaining storage for the UNAUTH output block.
Internal error	X'0000002C'	X'C1000001'	Error attempting to start RECON multiple update processing.
	X'0000002C'	X'C1000002'	Error attempting to end RECON multiple update processing.
	X'0000002C'	X'C1000006'	Entry in UNAUTH output block could not be found. This should not occur.
	X'0000002C'	X'C1000007'	Internal error encountered during DBRC unauthorization processing.
	X'0000002C'	X'C1000008'	Internal error encountered during DBRC unauthorization processing–invalid parameters.
Parameter error	X'00000030'	X'C1000001'	No AUTHLIST passed.
	X'00000030'	X'C1000002'	AUTHLIST passed with no entries.
	X'00000030'	X'C1000003'	Duplicate elements in AUTHLIST.

Table 100. DSPAPI FUNC=UNAUTH request return and reason codes (continued)

Code type	Return code	Reason code	Meaning
	X'00000030'	X'C1000004'	Missing or invalid OUTPUT parameter.
	X'00000030'	X'C9000001'	The function (FUNC) specified in the parameter list passed to the API is invalid.
	X'00000030'	X'C9000002'	Invalid TOKEN address. The address of the field containing the API TOKEN failed validity checking. The address specifies storage not owned by the calling program
	X'00000030'	X'C9000003'	Invalid RETCODE address. The address of the field containing the API RETCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000004'	Invalid RSNCODE address. The address of the field containing the API RSNCODE failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C9000005'	Invalid OUTPUT address. The address of the field containing the API OUTPUT failed validity checking. The address specifies storage not owned by the calling program.
	X'00000030'	X'C900000A'	An incorrect VERSION value was specified for the requested function (FUNC).
	X'00000030'	X'C900001A'	Invalid AUTHLIST address. The address of the field containing the API AUTHLIST failed validity checking. The address specifies storage not owned by the calling program.

Related reference
DBRC request return codes (Messages and Codes)

APAUB_RsnCode for UNAUTH output block

You can use this table to search for APAUB_RsnCode values for UNAUTH request return and reason codes. Each code is accompanied by an explanation of the code.

When an UNAUTH output block (DSPAPAUB) is returned, one of the following reason codes will be set in field APAUB_RsnCode for each element in the list of DBs or Areas in the request.

Table 101. APAUB_RsnCode values for UNAUTH request return and reason codes

APAUB_RsnCode	Meaning
X'00000000'	Request completed successfully.
X'C1000100'	Security error. SAF or the DBRC command authorization exit (DSPDCAX0) has determined that the user is not authorized to perform the request for this database or area.
X'C1000404'	Fast Path area needs recovery. It has been unauthorized.
X'C1000408'	Database not registered in RECON.
X'C1000410'	Subsystem not authorized to use the database.

Table 101. APAUB_RsnCode values for UNAUTH request return and reason codes (continued)

APAUB_RsnCode	Meaning
X'C1000414'	One of the following actions occurred: • Internal DBRC unauthorization error – Database and subsystem records do not match. • Previously active subsystem calling for unauthorization after a takeover occurred.
X'C1000418'	Internal DBRC or IMS unauthorization error– Held state could not be computed by the IMS compatibility evaluation routine.

Related reference

"DBRC unauthorization request (UNAUTH)" on page 445
You can use the UNAUTH request to explicitly remove authorization to a database or area. Authorization by an application is implicitly removed by the STOPDBRC request. UNAUTH is the opposite of FUNC=AUTH.

UNAUTH output block mapping

This figure illustrates the format of the output from a TYPE=UNAUTH request. The output block for the TYPE=UNAUTH request begins with a standard header that is mapped by the DSPAPQHD. The data portion of this output block is mapped by DSPAPAUB.

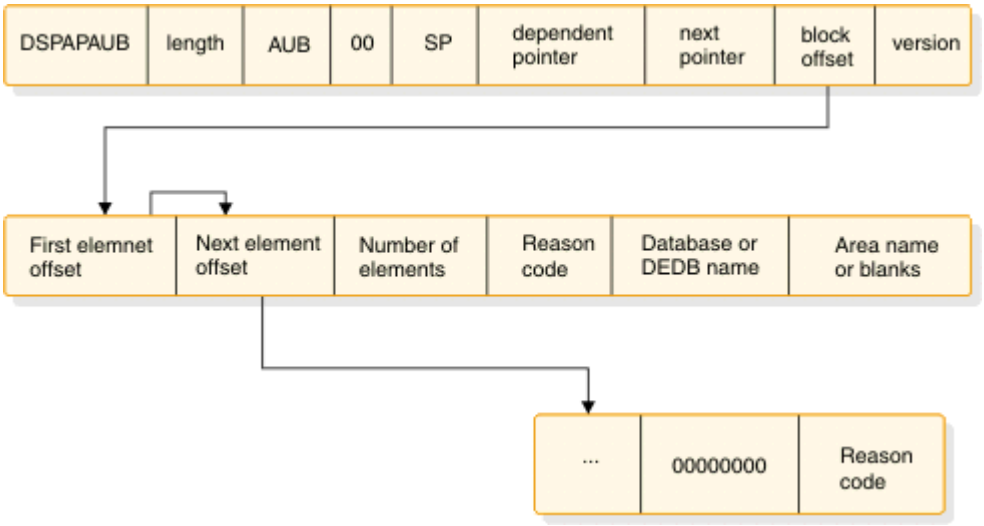


Figure 28. Format for a TYPE=UNAUTH output

UNAUTH output block

This figure illustrates the output block that is returned by the UNAUTH request. The output block contains an array of authorized databases and indicates if the UNAUTH request was successful.

Example of output block mapped by the DSPAPAUB

```
********************************************************************************
DSPAPAUB

  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)     DESCRIPTION
======== ======== ========= ========  ============== ==========================
       0      (0) STRUCTURE        8  DSPAPAUB       AUTH/UNAUTH block
       0      (0) UNSIGNED         4   APAUB_OFFSET  Offset to first element
       4      (4) SIGNED           4   APAUB_ELCOUNT Number of elements in list
```

```
  OFFSET   OFFSET
 DECIMAL     HEX  TYPE        LENGTH  NAME (DIM)     DESCRIPTION
======== ======== ========= ========  ============== ==========================
       0      (0) STRUCTURE       24  APAUB_ELEMENT
       0      (0) UNSIGNED         4   APAUB_OFFNEXT Offset to next element
       4      (4) SIGNED           4   APAUB_RSNCODE Reason code
       8      (8) CHARACTER        8   APAUB_DBNAME  Database or DEDB name
      16     (10) CHARACTER        8   APAUB_AREANAME
                                                     Area name or blanks

CONSTANTS

  LEN  TYPE           VALUE           NAME            DESCRIPTION
=====  =========  ================  ============== ==========================
    8  CHARACTER   DSPAPAUB         APAUB_EYECATCHER
```

Part 6. IMS catalog API (DFS3CATQ)

You can find information about using the IMS catalog API to request information about runtime application control blocks and program specification blocks from the IMS catalog.

Accessing the DFS3CATQ API

The DFS3CATQ API is provided with IMS in the DFS3CATQ assembler language macro.

Programming requirements

The DFS3CATQ API macro can be invoked from AMODE 24 or 64 callers. The invoking module can reside anywhere below the 2 GB bar.

Note: RMODE(64) is not supported.

The program that invokes the DFS3CATQ macro must be in task mode and not in cross-memory mode. The calling module need not reside in an APF-authorized library.

Execution environment

Programs that access the DFS3CATQ API can execute in an address space outside of IMS, and the IMS system need not be in the running state. If called from outside of IMS, an IMS RESLIB must be available in the standard z/OS search order of load modules.

Register usage

Input register information:

- Before invoking the DFS3CATQ macro, GPR 13 must point to a standard 18-word save area.

Output register information:

R0
Reason code

R1
Used as a work register

R2 - R13
Unchanged

R14
Used as a work register

R15
Return code

Related concepts

IMS catalog (Database Administration)

Related reference

IMS catalog utilities (System Utilities)

Chapter 37. IMS catalog API (DFS3CATQ macro)

Your application programs can retrieve database and program specification block definitions from the IMS catalog through the IMS catalog application programming interface (API), a release-independent assembler macro interface.

The IMS catalog API is provided with IMS in the DFS3CATQ macro.

To write a program that uses the IMS catalog API, you must have a working knowledge of the following subjects:

- Assembler language programming
- z/OS and the services it supplies
- IMS
- The IMS catalog

The DFS3CATQ macro supports the following actions:

- Acquire resources as needed to read the IMS catalog
- Get an object definition and return the information back to the calling application, optionally limiting the result to objects that match a name pattern
- List the objects that exist within the IMS catalog, optionally limiting the result to objects that match a name pattern
- Release resources held across invocations of the API

Related concepts

IMS catalog (Database Administration)

IMS management of ACBs (System Definition)

Related reference

IMS catalog utilities (System Utilities)

Chapter 38. Structure of applications that access the IMS catalog API

Your applications that use the IMS catalog API must use a certain general structure.

An application that uses the IMS catalog API has the following general structure:

1. Include the API DSECTs. (DFS3CATQ FUNC=DSECT)
2. Optionally, acquire the high-level qualifier (HLQ) of the boot strap data set (BSDS), only if the application is not aware of it.

   Note: The HLQ is to be provided before the DFS3CATQ FUNC=OPEN request is used. If the application is already aware of the HLQ of the BSDS, then it is to be provided on the DFS3CATQ FUNC=OPEN request.
3. Allocate and open the BSDS data set and directory data sets. (DFS3CATQ FUNC=OPEN)
4. Request information from the IMS catalog by using one or more of either of the following actions:
   - Request a list of object names that match a specified name and type. (DFS3CATQ FUNC=LIST)
   - Request information about the definition of a specified catalog object by specifying the object name and type. (DFS3CATQ FUNC=GET)
5. Release data sets that were allocated by the open request. (DFS3CATQ FUNC=CLOSE)

Chapter 39. DSECT mapping request (DSECT) for the IMS catalog API

You can use an IMS catalog API mapping request to build a DSECT for mapping the returned block of storage.

Syntax for DSECT requests

Chapter 40. HLQ request (HLQ) for the IMS catalog API

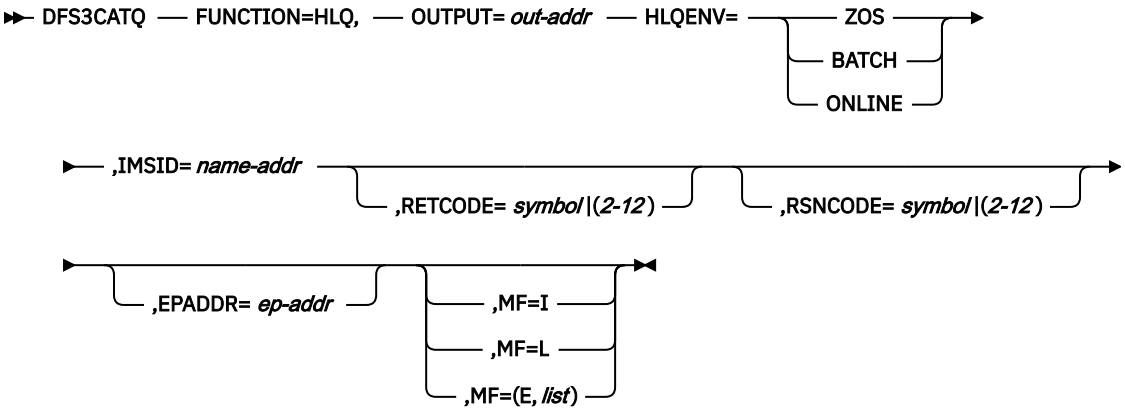
Use the HLQ function of the IMS catalog API to acquire the high-level qualifier (HLQ) for the boot strap data set (BSDS) name for the subsequent IMS catalog API OPEN requests that require the BSDS name.

During the HLQ request, the IMS directory boot strap data set (BSDS) is found and returned in the output area. The BSDS information can then be saved and provided to the API OPEN request.

When you make a HLQ request, the DFS3CATQ macro must specify the environment in which the request is executed. You can use the HLQ request to find BSDS in the following environment:

- IMS online regions of type BMP or IFP.
- IMS batch regions of type DLI, ULU, or DBB.
- z/OS environment where an IMS control region is available on the same LPAR.

Syntax for HLQ requests



Example

```
DFS3CATQ FUNCTION=HLQ,OUTPUT=(R2),HLQENV=ZOS,
    IMSID=(R3)
```

Parameters for HLQ requests

EPADDR=*ep-addr* | RS-type address, or register (2-12).
If specified as a symbol, specifies the label of a word of storage that contains the address of the load module DFS3CATQ. The application is responsible for loading module DFS3CATQ, saving its entry point address for this parameter and deleting the load module when it is no longer needed.

HLQENV=
Specifies the environment where the HLQ request is executed.

BATCH
IMS batch regions of type ULU, DLI, and DBB.

ONLINE
IMS online regions of type BMP and IFP.

ZOS
Programs that are executed by z/OS. When you use the **HLQENV=ZOS** parameter, the IMS control region must be up.

IMSID=*name-addr* | RS-type address, or register (2-12)
Specifies the address of the 4-byte application storage area that identifies the IMS system for the associated BSDS. When more than one IMS system is present on an LPAR, the **IMSID=** parameter must be paired with the **HLQENV=ZOS** parameter.

MF=
The macro form of the request.

I
Invokes the DFSCATQx program with an in-line parameter list. If your program is reentrant, do not use this form of the macro because reentrant code cannot be modified.

L
Specifies the list form of the macro.

(E,*list*)
Specifies the execute form of the macro.

list | RS-type address, or register (2-12).

OUTPUT=*out-addr* | RS-type address, or register (2-12).
Specifies the address of a 4-byte field to receive the address of the first storage area that contains the information for the request.

Additional storage may be needed to contain the full set of information requested. Each additional area of storage is chained off of the prior one. The application is responsible for freeing the output areas when they are no longer needed.

RETCODE=*symbol* | (*2-12*)
Saves the return code to a storage location determined by the specified symbol or register.

If a symbol is specified, the symbol must represent the label of a word of storage in which to save the return code.

If a register is specified, the register must contain the address of a word of storage in which to save the return code. Specify a register from 2 to 12 that is enclosed in parentheses.

Regardless of whether RETCODE is specified, IMS returns the return code in register 15.

RSNCODE=*symbol* | (*2-12*)
Saves the reason code to a storage location determined by the specified symbol or register.

If a symbol is specified, the symbol must represent the label of a word of storage in which to save the reason code.

If a register is specified, the register must contain the address of a word of storage in which to save the reason code. Specify a register from 2 to 12 that is enclosed in parentheses.

Regardless of whether RSNCODE is specified, IMS returns the reason code in register 0.

Output area for HLQ requests

The returned storage of the HLQ request is consist of a 16-byte prefix and the HLQ of the BSDS required by the OPEN request.

With each successful HLQ request, DFS3CATQ acquires storage to hold the requested information. The address of the storage obtained is stored in the caller's area (see the OUTPUT= parameter).

The storage is allocated by the STORAGE macro with LOC=31 and SP=0.

More than one output area may be needed to contain all of the information requested. The address of the next output area is stored in the area. The application is responsible for freeing the output areas when they are no longer needed.

Each output area has the following format:

Table 102. Output area returned for HLQ requests

Content	Description
Size	4 bytes. The size of this storage area.
Address	4 bytes. The address of the next area, or '00000000'x if this is the last area.
Reserved	8 bytes. Reserved for internal use.
Data area	The requested data in the following format: **44 bytes** The Prefix of the IMS Boot strap data set. **4 bytes** The catalog alias Prefix.

Return and reason code for HLQ requests

Table 103. Return and reason codes for HLQ requests

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'0000000C'	X'0000000C'	Valid address for the OUTPUT= variable or location was not provided.
X'0000000C'	X'00000010'	Valid address for the TOKEN= variable or location was not provided.
X'0000000C'	X'00000200'	The HLQ request was unsuccessful due to a missing or invalid parameter.
X'00000014'	X'00000214'	The HLQ request was unsuccessful. The specified IMS control program is not active.
X'00000024'	X'00000204'	The HLQ request was unsuccessful using an internal service. The request was unable to get to the catalog anchor block.
X'00000024'	X'00000208'	The HLQ request was unsuccessful using an internal service. The request was unable to get to the catalog directory.
X'00000024'	X'0000020C'	The HLQ request was unsuccessful using an internal service. A severe error occurred.
X'00000024'	X'00000218'	A GSCD call was performed, and the address of the SCD was not returned.

Table 103. Return and reason codes for HLQ requests (continued)

Return Code	Reason Code	Meaning
X'aaaabbbb'	X'cccccccc'	An IMS or MVS service returned an unexpected return or reason code. aaaa identifies the service that was performed. bbbb is the service return code. cccccccc is the service reason code. The possible values for aaaa and the service each identifies are: • 0001 - ISGENQ • 0002 - IEANTCR • 0003 - IEANTRT For more information about the return and reason codes that may be returned for each service, refer to the following topics in the z/OS documentation: ISGENQ — Global resource serialization ENQ service IEANTCR — Create a name/token pair IEANTRT — Retrieve the token from a name/token pair

Related reference

“Open request (OPEN) for the IMS catalog API” on page 465
Use the IMS catalog API OPEN function to allocate either the IMS directory data set or the IMS directory staging data set for subsequent API calls to list or retrieve the database and program view resources that are defined to IMS.

Related information

IBM z/OS documentation

Chapter 41. Open request (OPEN) for the IMS catalog API

Use the IMS catalog API OPEN function to allocate either the IMS directory data set or the IMS directory staging data set for subsequent API calls to list or retrieve the database and program view resources that are defined to IMS.

During the OPEN request, the IMS directory boot strap data set (BSDS) is allocated, opened, read, closed, and unallocated. If you specify DEFINITION=CURRENT to retrieve the definitions that are currently active in the IMS system, the IMS directory data set remains allocated until the CLOSE request. If you specify DEFINITION=PENDING to retrieve any definitions that are waiting to be activated, the staging data set of the IMS directory remains allocated until the CLOSE request.

When you make an OPEN request, the DFS3CATQ macro dynamically allocates the BSDS and directory data sets as needed. It allocates a block of virtual storage that is used to communicate information across subsequent GET and LIST requests. The address of the block is stored at the address that is provided in the TOKEN parameter. The same token must be used for subsequent requests.

You can issue a CLOSE request to free the allocation.

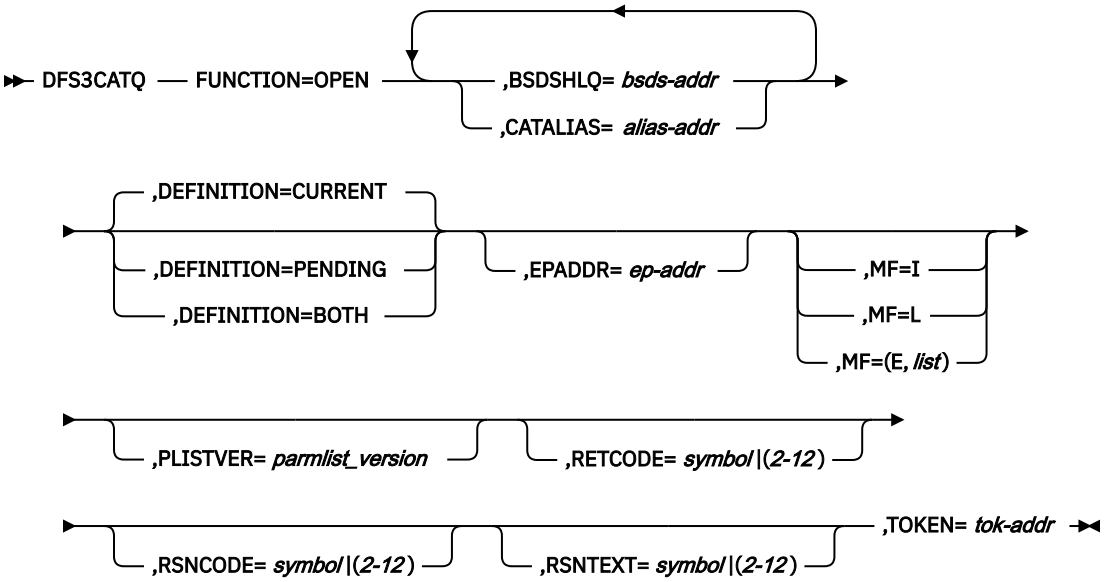
You can make multiple IMS catalog API OPEN requests and allow them to remain in use before you issue a CLOSE request for any specified OPEN request.

To have multiple IMS catalog API allocations active at the same time:

1. Provide a different TOKEN address for each OPEN request.
2. Specify that token for each subsequent GET or LIST request against that IMS catalog.
3. Specify that token on the CLOSE request to free the allocation of that IMS directory data set.

Up to 36 OPEN requests can be active at one time if the system-generated DDname is available and there are no environment limits.

Syntax for OPEN requests



Parameters for OPEN requests

BSDSHLQ=*bsds-addr* | RS-type address, or register (2-12).
The address of an area in application storage that contains the high-level qualifier prefix for the IMS boot strap data set. The first two bytes of storage must contain the length of the high-level qualifier prefix that follows.

The qualifier '.BSDS' is appended to the high-level qualifier prefix value to form the full name of the IMS boot strap data set.

If both BSDSHLQ and CATALIAS are specified, IMS locates the IMS directory by using the high-level qualifier that is defined in the CATDSHLQ DFSMDA member that corresponds to the CATALIAS value. The DFSMDA member must exist either in a library allocated as //IMSDALIB, or in the //JOBLIB or // STEPLIB libraries. The IMS catalog API first searches in //IMSDALIB, if it exists, for the MDA members. If IMS catalog API does not find the DFSMDA member, the IMS catalog API uses the high-level qualifier at the BSDSHLQ location.

CATALIAS=*alias-addr*

The address of an area in application storage that contains the 4-character IMS catalog alias name that is specified on the DDNAME parameter of the CATDSHLQ DFSMDA macro statement that contains the high-level qualifier of the IMS directory data sets.

If both BSDSHLQ and CATALIAS are specified, IMS locates the IMS directory by using the high-level qualifier that is defined in the CATDSHLQ DFSMDA member that corresponds to the CATALIAS value. The DFSMDA member must exist either in a library allocated as //IMSDALIB, or in the //JOBLIB or // STEPLIB libraries. The IMS catalog API first searches in //IMSDALIB, if it exists, for the MDA members. If IMS catalog API does not find the DFSMDA member, the IMS catalog API uses the high-level qualifier at the BSDSHLQ location.

If a symbol is specified, you must specify RSNTEXT.

DEFINITION=*definition-type*

BOTH
The definitions of any resources in the CURRENT or PENDING data sets of the IMS directory are returned or listed.

CURRENT
The definitions of the resources that are currently active in the IMS system are returned or listed. This is the default.

PENDING
The definitions of any resources in the staging data set of the IMS directory that are pending activation are returned or listed.

EPADDR=*ep-addr* | RS-type address, or register (2-12).
If specified as a symbol, specifies the label of a word of storage that contains the address of the load module DFS3CATQ. The application is responsible for loading module DFS3CATQ, saving its entry point address for this parameter and deleting the load module when it is no longer needed.

MF=
The macro form of the request.

I
Invokes the DFSCATQx program with an in-line parameter list. If your program is reentrant, do not use this form of the macro because reentrant code cannot be modified.

L
Specifies the list form of the macro.

(E,*list*)
Specifies the execute form of the macro.

list | RS-type address, or register (2-12).

PLISTVER=*parmlist_version*
The version of the OPEN function parameter list.

1
Specifies version 1 of the parameter list for the OPEN function.

Version 1 is the default.

2
Specifies version 2 of the parameter list for the OPEN function.

Version 2 of the parameter list includes the **CATALIAS**, **PLISTVER**, and **RSNTEXT** parameters and is larger in size than version 1.

1

RETCODE=*symbol* | (*2-12*)
Saves the return code to a storage location determined by the specified symbol or register.

If a symbol is specified, the symbol must represent the label of a word of storage in which to save the return code.

If a register is specified, the register must contain the address of a word of storage in which to save the return code. Specify a register from 2 to 12 that is enclosed in parentheses.

Regardless of whether RETCODE is specified, IMS returns the return code in register 15.

RSNCODE=*symbol* | (*2-12*)
Saves the reason code to a storage location determined by the specified symbol or register.

If a symbol is specified, the symbol must represent the label of a word of storage in which to save the reason code.

If a register is specified, the register must contain the address of a word of storage in which to save the reason code. Specify a register from 2 to 12 that is enclosed in parentheses.

Regardless of whether RSNCODE is specified, IMS returns the reason code in register 0.

RSNTEXT=*symbol* | (*2-12*)
Saves the reason text to a storage location determined by the specified symbol or register. The reason text storage area must be defined with 120 bytes long.

If a symbol is specified, the symbol must represent the label of a word of storage in which to save the reason text.

If a register is specified, the register must contain the address of a word of storage in which to save the reason text. Specify a register from 2 to 12 that is enclosed in parentheses.

If a symbol is specified, you must specify CATALIAS.

TOKEN=*tok-addr* | RS-type address, or register (2-12).
Specifies the address of a 4-byte field to receive the API token. Your program receives this token when a DFS3CATQ FUNC=OPEN macro is issued. This token must be supplied with all other macro calls that are associated with this instance of the OPEN request. The token is no longer valid after a DFS3CATQ FUNC=CLOSE macro call.

Return and reason codes

Table 104. Return and reason codes for the DFS3CATQ macro OPEN requests

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'0000000C'	X'00000004'	Request block level error.
X'0000000C'	X'00000008'	BSDS qualifier length error.

Table 104. Return and reason codes for the DFS3CATQ macro OPEN requests (continued)

Return Code	Reason Code	Meaning
X'0000000C'	X'0000000C'	A valid address for the OUTPUT= variable or location was not provided.
X'0000000C'	X'00000010'	The address provided for the TOKEN= variable or location was not valid.
X'0000000C'	X'00000020'	The address provided for the RSNTEXT= variable or location was not valid.
X'00000014'	X'*yyyyzzzz*'	The OPEN request was unsuccessful using an internal service. The reason code contains the service's return code (*yyyy*) and reason code (*zzzz*).
X'00000018'	X'*nnnnnnnn*'	OPEN was not successful. The reason code is the return code from the OPEN macro.
X'00000024'	X'0000002C'	The OPEN request failed due to no available DDname for the allocation.
X'00000024'	X'00000030'	The token area contains invalid data.
X'0000002C'	X'*nnnnnnnn*'	OPEN boot strap data set (BSDS) in output mode was not successful. The reason code is the return code from the OPEN macro.
X'00000030'	X'*yyyyzzzz*'	Dynamic allocation of IMS directory data set was not successful. The reason code contains S99ERROR (yyyy) and S99INFO (zzzz) from the DYNALLOC macro.
X'00000034'	X'*yyyyzzzz*'	Dynamic deallocation of boot strap data set (BSDS) was not successful. The reason code contains S99ERROR (yyyy) and S99INFO (zzzz) from the DYNALLOC macro.
X'00000038'	X'*yyyyzzzz*'	Dynamic allocation of concatenated directory data set was not successful. The reason code contains S99ERROR (yyyy) and S99INFO (zzzz) from the DYNALLOC macro.

Table 104. Return and reason codes for the DFS3CATQ macro OPEN requests (continued)

Return Code	Reason Code	Meaning
X'0000003C'	X'00000024'	The member specified with the CATALIAS= parameter was not found.
X'0000003C'	X'00000028'	An error occurred when the LOAD macro attempted to load the member specified with the CATALIAS= parameter.
X'0000003C'	X'00000104'	The contents of the DFSMDA specified with the CATALIAS= parameter are invalid.
X'0000003C'	X'00000108'	Storage for the DFSMDA in IMSDALIB could not be obtained.
X'00000044'	X'*yyyyzzzz*'	Dynamic allocation of boot strap data set (BSDS) was not successful. The reason code contains S99ERROR (yyyy) and S99INFO (zzzz) from the DYNALLOC macro.

Related reference

“ Close request (CLOSE) for the IMS catalog API” on page 481

You can use an IMS catalog API CLOSE request to close any data sets that were allocated for previous IMS catalog API requests.

Chapter 42. Get request (GET) for the IMS catalog API

Use the GET function of the IMS catalog API to get an object definition from the IMS catalog and return the information back to the calling application.

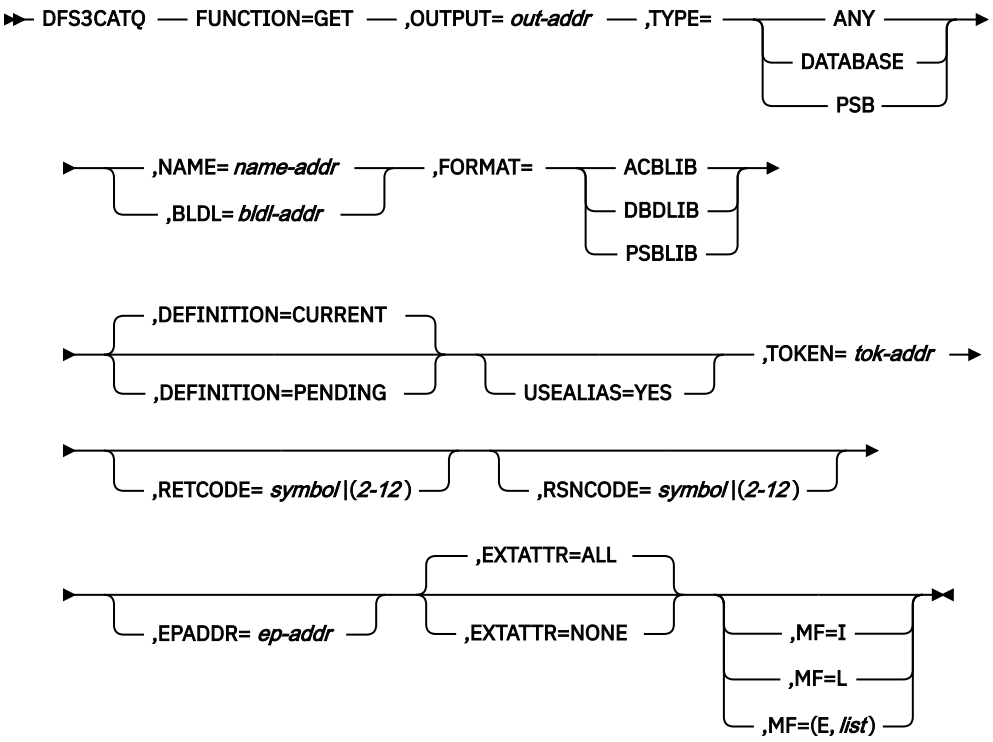
For each successful GET request, one or more areas of storage is returned to the application that contains the requested information. The address of the first area of storage is stored at the address provided in the OUTPUT parameter. The address of the next area of storage, if any, is contained within the first area. For the last area, the address of the next area of storage is null.

The application is responsible for freeing each of these storage areas when they are no longer needed.

Restriction:

GSAM ACB or DBD control blocks cannot be retrieved from the IMS catalog by using the GET request of the IMS catalog API.

Syntax for GET requests



Parameters for GET requests

BLDL=*bldr-addr* | RS-type address, or register (2-12).
The address of an area in application storage where a list of object names resides. See the BLDL macro and its list-address parameter of the BLDL macro for the format and requirements for this area.

DEFINITION=*definition-type*
If DEFINITION=BOTH is specified for the OPEN request, the GET request must specify which data set of the IMS catalog, staging or directory, to retrieve information from:

CURRENT
The definitions of the resources that are currently active in the IMS system, that is, the definitions of the resources that are in the directory data set of the IMS catalog, are returned. This is the default value.

PENDING
The definitions of any resources that are pending activation, that is, the definitions of the resources that are in the staging data set of the IMS catalog, are returned.

EXTATTR=*output-extended-attribute-type*
Determines whether extended attribute metadata is returned. This parameter is valid only for the **FORMAT=DBDLIB** parameter.

ALL
All extended attribute metadata is returned. This is the default value.

NONE
Only the vendor section table is returned. If the vendor section table does not exist, no extended attribute metadata is not returned.

EPADDR=*ep-addr* | RS-type address, or register (2-12).
If specified as a symbol, specifies the label of a word of storage that contains the address of the load module DFS3CATQ. The application is responsible for loading module DFS3CATQ, saving its entry point address for this parameter and deleting the load module when it is no longer needed.

FORMAT=*format-type*
The format in which to return the requested object definition.

ACBLIB
Return in ACBLIB format.

DBDLIB
Return in DBDLIB format.

PSBLIB
Return in PSBLIB format.

MF=
The macro form of the request.

I
Invokes the DFSCATQx program with an in-line parameter list. If your program is reentrant, do not use this form of the macro because reentrant code cannot be modified.

L
Specifies the list form of the macro.

(E,*list*)
Specifies the execute form of the macro.

list | RS-type address, or register (2-12).

NAME=*name-addr* | RS-type address, or register (2-12).
The address of an area in application storage describing the name or name pattern of the object to retrieve. The first 2 bytes of storage should contain the length of the name or pattern that follows.

A naming pattern can be provided by specifying a wild card mask. An asterisk can represent zero, one, or more characters in the name. A percent sign represents exactly one character in the name. To get all objects, specify only an asterisk.

OUTPUT=*out-addr* | RS-type address, or register (2-12).
Specifies the address of a 4-byte field to receive the address of the first storage area that contains the information for the request.

Additional storage may be needed to contain the full set of information requested. Each additional area of storage is chained off of the prior one. The application is responsible for freeing the output areas when they are no longer needed.

RETCODE=*symbol* | (*2-12*)
Saves the return code to a storage location determined by the specified symbol or register.

If a symbol is specified, the symbol must represent the label of a word of storage in which to save the return code.

If a register is specified, the register must contain the address of a word of storage in which to save the return code. Specify a register from 2 to 12 that is enclosed in parentheses.

Regardless of whether RETCODE is specified, IMS returns the return code in register 15.

RSNCODE=*symbol* | (*2-12*)
Saves the reason code to a storage location determined by the specified symbol or register.

If a symbol is specified, the symbol must represent the label of a word of storage in which to save the reason code.

If a register is specified, the register must contain the address of a word of storage in which to save the reason code. Specify a register from 2 to 12 that is enclosed in parentheses.

Regardless of whether RSNCODE is specified, IMS returns the reason code in register 0.

TOKEN=*tok-addr* | RS-type address, or register (2-12).
Specifies the address of a 4-byte field to receive the API token. Your program receives this token when a DFS3CATQ FUNC=OPEN macro is issued. This token must be supplied with all other macro calls that are associated with this instance of the OPEN request. The token is no longer valid after a DFS3CATQ FUNC=CLOSE macro call.

TYPE=*object-type*
The type of object that is requested, where *object-type* is one of the following values:

ANY
Any object type is requested.

DATABASE
Database types are requested.

PSB
Program specification block types are requested.

USEALIAS=YES
Returns the IMS catalog database with its alias name in the output area. This requires that a FUNCTION=HLQ or FUNCTION=OPEN,CATALIAS= call was used prior to this request.

Output area for GET requests

With each successful GET request or X'00000028' - X'00000006' (DCPLRC_INVSEG) return-reason code combination (see the X'00000028' return code row in Table 106 on page 474 for details), DFS3CATQ acquires storage to hold the requested information. The address of the storage obtained is stored in the caller's area (see the OUTPUT= parameter). The storage is allocated by the STORAGE macro with LOC=31 and SP=0.

More than one output area might be needed to contain all of the information requested. The address of the next output area is stored in the area.

The application is responsible for freeing the output areas when they are no longer needed.

Each output area has the following format:

Table 105. Output area returned for GET requests

Content	Description
Size	4 bytes. The size of this storage area.
Address	4 bytes. The address of the next area, or '00000000'x if this is the last area.

Table 105. Output area returned for GET requests (continued)

Content	Description
Reserved	1 byte. Reserved for internal use.
Flags	1 byte. X`80`: EXTATTR=NONE is specified.
Size	2 bytes containing the size of each data area.
Reserved	4 bytes. Reserved for internal use.
Data area	The requested data is comprised of the following repeating elements: **4 bytes** The address of the object. The object is mapped by one of the following macros: • IDBD for a DBD returned in the DBDLIB format. • DFSDMB for a DBD returned in the ACBLIB format. • DFSIPSB for a PSB returned in the PSBLIB format. • DFSPSB for a PSB returned in the ACBLIB format. **4 bytes** Length of the object. **8 bytes** Member name of the object.

Return and reason codes for GET requests

Table 106. Return and reason codes for the DFS3CATQ macro

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000004'	X'00000000'	One or more list entries in the area specified by using the **BLDL=***parameter* could not be found. Any entries that contain a wild card mask could not be found.
X'0000000C'	X'0000000C'	A valid address for the **OUTPUT=***variable or location* was not provided.
X'0000000C'	X'00000010'	A valid address for the **TOKEN=***variable or location* was not provided.
X'0000000C'	X'00000018'	One of the parameter pairs is specified: • **DEFINITION=CURRENT** for the OPEN request and **DEFINITION=PENDING** for the GET request. • **DEFINITION=PENDING** for the OPEN request and **DEFINITION=CURRENT** for the GET request.

Table 106. Return and reason codes for the DFS3CATQ macro (continued)

Return Code	Reason Code	Meaning
X'00000028'	X'*nnnnnnnn*'	An attempt by the GET request to use an internal service was unsuccessful. The return code contains the service's return code (*nnnnnnnn*). The reason code *nnnnnnnn* can be from the DFS3CPL return codes. The associated labels and meaning are explained in the following list: **X'00000001' (DCPLRC_SIZE)** Insufficient output buffer size. **X'00000002' (DCPLRC_INVMBR)** Invalid DBD or PSB member. **X'00000003' (DCPLRC_VIRSEG)** Invalid virtually paired segment name. **X'00000004' (DCPLRC_SECIDX)** Invalid secondary index name. **X'00000005' (DCPLRC_INVDBD)** Invalid DBD name. **X'00000006' (DCPLRC_INVSEG)** Invalid segment name.* *The API will generate output for this error-reason code combination. The application is responsible for freeing the output areas when they are no longer needed. **X'00000007' (DCPLRC_BLDERR)** Error building catalog segments. **X'00000008' (DCPLRC_INVACB)** Invalid ACBLIB member. **X'00000009' (DCPLRC_ISAM)** Invalid ISAM. **X'0000000A' (DCPLRC_ACBERR)** Error processing ACBLIB member. **X'0000FFFF' (DCPLRC_UNKERR)** Unknown error.
X'0000002C'	X'00000034'	Storage for the decoder output area could not be obtained.
X'00080004'	X'*yyyyzzzz*'	The request was unsuccessful during BLDL macro processing. The reason code contains the BLDL return code (*yyyy*) and reason code (*zzzz*).
X'00080008'	X'*yyyyzzzz*'	The request was unsuccessful during FIND macro processing. The reason code contains the FIND return code (*yyyy*) and reason code (*zzzz*).

Related reference

Writing IMS routines that access control blocks (Exit Routines)

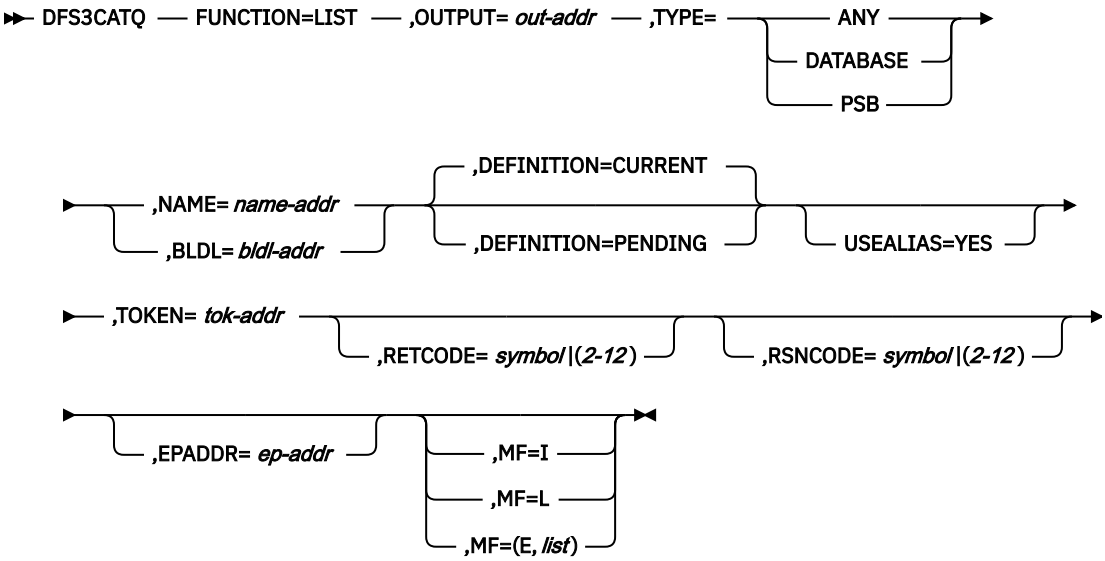
Chapter 43. List request (LIST) for the IMS catalog API

You can use the IMS catalog API list request to return a list of object names that match a specified object type and name.

For each successful LIST request, one or more areas of storage is returned to the application that contains the requested information. The address of the first area of storage is stored at the address provided in the OUTPUT parameter. The address of the next area of storage, if any, is contained within the first area. For the last area, the address of the next area of storage is null.

The application is responsible for freeing each of these storage areas when they are no longer needed.

Syntax for LIST requests



Parameters for LIST requests

BLDL=*bldr-addr* | RS-type address, or register (2-12).
The address of an area in application storage where a list of object names resides. See the BLDL macro and its list-address parameter of the BLDL macro for the format and requirements for this area.

If this parameter is specified, the **TYPE=** parameter is ignored.

DEFINITION=*definition-type*
If DEFINITION=BOTH is specified for the OPEN request, the LIST request must specify which data set of the IMS catalog, staging or directory, to retrieve information from:

CURRENT
The definitions of the resources that are currently active in the IMS system, that is, the definitions of the resources that are in the directory data set of the IMS catalog, are listed. This is the default value.

PENDING
The definitions of any resources that are pending activation, that is, the definitions of the resources that are in the staging data set of the IMS catalog, are listed.

EPADDR=*ep-addr* | RS-type address, or register (2-12).
If specified as a symbol, specifies the label of a word of storage that contains the address of the load module DFS3CATQ. The application is responsible for loading module DFS3CATQ, saving its entry point address for this parameter and deleting the load module when it is no longer needed.

MF=
The macro form of the request.

I
Invokes the DFSCATQx program with an in-line parameter list. If your program is reentrant, do not use this form of the macro because reentrant code cannot be modified.

L
Specifies the list form of the macro.

(E,*list*)
Specifies the execute form of the macro.

list | RS-type address, or register (2-12).

NAME=*name-addr*
RS-type address, or register (2) - (12).

The name pattern to use for the LIST request. The first two bytes of storage should contain the length of the name or pattern that follows. The naming pattern can be provided by specifying a wild card mask. An asterisk can represent zero, one, or more characters in the name. A percent sign represents exactly one character in the name. To list all objects, specify only an asterisk.

OUTPUT=*out-addr* | RS-type address, or register (2-12).
Specifies the address of a 4-byte field to receive the address of the first storage area that contains the information for the request.

Additional storage may be needed to contain the full set of information requested. Each additional area of storage is chained off of the prior one. The application is responsible for freeing the output areas when they are no longer needed.

RETCODE=*symbol* | (*2-12*)
Saves the return code to a storage location determined by the specified symbol or register.

If a symbol is specified, the symbol must represent the label of a word of storage in which to save the return code.

If a register is specified, the register must contain the address of a word of storage in which to save the return code. Specify a register from 2 to 12 that is enclosed in parentheses.

Regardless of whether RETCODE is specified, IMS returns the return code in register 15.

RSNCODE=*symbol* | (*2-12*)
Saves the reason code to a storage location determined by the specified symbol or register.

If a symbol is specified, the symbol must represent the label of a word of storage in which to save the reason code.

If a register is specified, the register must contain the address of a word of storage in which to save the reason code. Specify a register from 2 to 12 that is enclosed in parentheses.

Regardless of whether RSNCODE is specified, IMS returns the reason code in register 0.

TOKEN=*tok-addr* | RS-type address, or register (2-12).
Specifies the address of a 4-byte field to receive the API token. Your program receives this token when a DFS3CATQ FUNC=OPEN macro is issued. This token must be supplied with all other macro calls that are associated with this instance of the OPEN request. The token is no longer valid after a DFS3CATQ FUNC=CLOSE macro call.

TYPE=*object-type*
The type of object that is requested, where *object-type* is one of the following values:

ANY
Any object type is requested.

DATABASE
Database types are requested.

PSB
Program specification block types are requested.

If the **BLDL=** parameter is specified, this parameter is ignored.

USEALIAS=YES
Returns the IMS catalog database with its alias name in the output area. This requires that a FUNCTION=HLQ or FUNCTION=OPEN,CATALIAS= call was used prior to this request.

Output area for LIST requests

With each successful LIST request, DFS3CATQ acquires storage to hold the requested information. The address of the storage obtained is stored in the caller's area (see the OUTPUT= parameter). The storage is allocated by the STORAGE macro with LOC=31 and SP=0.

More than one output area may be needed to contain all of the information requested. The address of the next output area is stored in the area.

The application is responsible for freeing the output areas when they are no longer needed.

Each output area has the following format:

Table 107. Output area returned for LIST requests

Content	Description
Size	4 bytes. The size of this storage area.
Address	4 bytes. The address of the next area, or '00000000'x if this is the last area.
Reserved	8 bytes. Reserved for internal use.
Data area	The requested data, in the following format: **2 bytes** The number of members returned. **2 bytes** Length of information (*nn*) returned for each member. ***nn* bytes** Member information, mapped by the IMS ACBDIR PREFIX=BLDL macro.

Return and reason codes for LIST requests

Table 108. Return and reason codes for the DFS3CATQ macro LIST requests

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'00000004'	X'00000000'	One or more list entries in the area specified by using the **BLDL=** parameter could not be found.

Table 108. Return and reason codes for the DFS3CATQ macro LIST requests (continued)

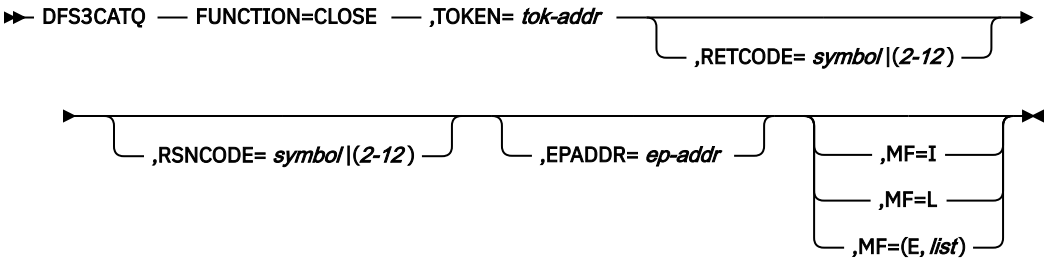
Return Code	Reason Code	Meaning
X'0000000C'	X'0000000C'	A valid address for the OUTPUT= variable or location was not provided.
X'0000000C'	X'00000010'	Valid address for the TOKEN= variable or location was not provided.
X'0000000C'	X'00000018'	One of the parameter pairs is specified: • DEFINITION=CURRENT for the OPEN request and DEFINITION=PENDING for the GET request • DEFINITION=PENDING for the OPEN request and DEFINITION=CURRENT for the GET request
X'0000000C'	X'00000100'	Invalid value for the FUNCTION parameter.
X'00080004'	X'yyyyzzzz'	The request was unsuccessful during BLDL macro processing. The reason code contains the BLDL return code (*yyyy*) and reason code (*zzzz*).
X'00080008'	X'yyyyzzzz'	The request was unsuccessful during FIND macro processing. The reason code contains the FIND return code (*yyyy*) and reason code (*zzzz*).

Chapter 44. Close request (CLOSE) for the IMS catalog API

You can use an IMS catalog API CLOSE request to close any data sets that were allocated for previous IMS catalog API requests.

When you issue a CLOSE request you free any data sets that were allocated for the communication area by an OPEN request, or for subsequent GET and LIST requests.

Syntax for CLOSE requests



Parameters for CLOSE requests

TOKEN=*tok-addr* | RS-type address, or register (2-12).
Specifies the address of a 4-byte field to receive the API token. Your program receives this token when a DFS3CATQ FUNC=OPEN macro is issued. This token must be supplied with all other macro calls that are associated with this instance of the OPEN request. The token is no longer valid after a DFS3CATQ FUNC=CLOSE macro call.

RETCODE=*symbol* | (*2-12*)
Saves the return code to a storage location determined by the specified symbol or register.

If a symbol is specified, the symbol must represent the label of a word of storage in which to save the return code.

If a register is specified, the register must contain the address of a word of storage in which to save the return code. Specify a register from 2 to 12 that is enclosed in parentheses.

Regardless of whether RETCODE is specified, IMS returns the return code in register 15.

RSNCODE=*symbol* | (*2-12*)
Saves the reason code to a storage location determined by the specified symbol or register.

If a symbol is specified, the symbol must represent the label of a word of storage in which to save the reason code.

If a register is specified, the register must contain the address of a word of storage in which to save the reason code. Specify a register from 2 to 12 that is enclosed in parentheses.

Regardless of whether RSNCODE is specified, IMS returns the reason code in register 0.

EPADDR=*ep-addr* | RS-type address, or register (2-12).
If specified as a symbol, specifies the label of a word of storage that contains the address of the load module DFS3CATQ. The application is responsible for loading module DFS3CATQ, saving its entry point address for this parameter and deleting the load module when it is no longer needed.

MF=
The macro form of the request.

I
Invokes the DFSCATQx program with an in-line parameter list. If your program is reentrant, do not use this form of the macro because reentrant code cannot be modified.

L
Specifies the list form of the macro.

(E,*list*)
Specifies the execute form of the macro.

list | RS-type address, or register (2-12).

Return and reason codes for CLOSE requests

Table 109. Return and reason codes for the DFS3CATQ macro OPEN requests

Return Code	Reason Code	Meaning
X'00000000'	X'00000000'	Request completed successfully.
X'0000000C'	X'0000000C'	A valid address for the OUTPUT= variable or location was not provided.
X'0000000C'	X'00000010'	Valid address for the TOKEN= variable or location was not provided.

Related reference
“Open request (OPEN) for the IMS catalog API” on page 465
Use the IMS catalog API OPEN function to allocate either the IMS directory data set or the IMS directory staging data set for subsequent API calls to list or retrieve the database and program view resources that are defined to IMS.

Part 7. IMS installed level API (DFSGVRM)

You can use the DFSGVRM API that is contained in the DFSGVRM macro in IMS to make a call from your applications to obtain the version, release, and modification level of an IMS system.

The DFSGVRM API can be called from within or outside of an IMS system with no specific authorization required for the call. When the call is made from within an IMS system, the IMS system must be in the running state. When the call is made from outside of an IMS system, the IMS system need not be in the running state.

The DFSGVRM macro supports the following functions:

FUNC=CALL
Gets the version, release, modification level of the IMS system and returns the information to an area in storage from where the calling application can retrieve the details.

FUNC=REL
Releases the storage used for the output that is requested by the calling application. The storage is released when it is no longer required. The calling application must use the **REL** function when the **RETAREA** register or the parameter list field **GVRMP_A_OUTPUT_AREA** is *nonzero*, regardless of the return code.

Accessing the DFSGVRM API

The DFSGVRM API is provided with IMS in the DFSGVRM assembler language macro.

Programming requirements

The DFSGVRM macro can be invoked from AMODE 24, 31, or 64 callers. The invoking module can reside anywhere below the 2 GB bar.

Note: RMODE(64) is not supported.

The program that invokes the DFSGVRM macro must be in task mode and not in cross-memory mode. The calling module need not reside in an APF-authorized library.

Execution environment

Programs that access the DFSGVRM API can execute in any IMS address space while the IMS system is in the running state.

Programs that access the DFSGVRM API can also execute in an address space outside of IMS, and the IMS system need not be in the running state. If called from outside of IMS, an IMS RESLIB must be available in the standard z/OS search order of load modules.

Register usage

Input register information:

- Before invoking the DFSGVRM macro, GPR 13 must point to a standard 18-word save area.

Output register information:

R0
Reason code

R1
Used as a work register

R2 - R13
Unchanged

R14
Used as a work register

R15
Return code

Chapter 45. CALL request (CALL) for the IMS installed level API

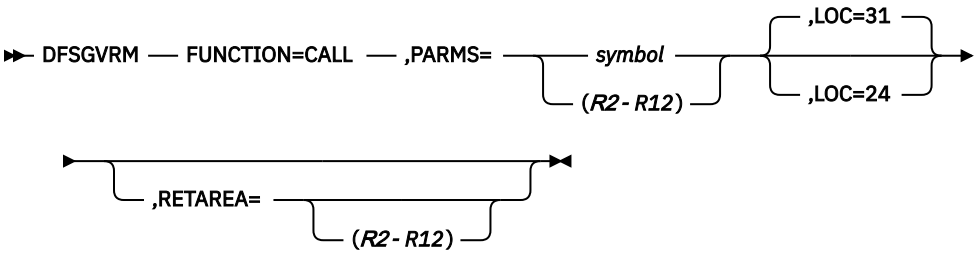
Use the CALL function of the DFSGVRM API to obtain the version, release, and modification level of the IMS system.

When the calling application issues the CALL request, IMS builds a parameter list to call module DFSGVRM0, and the parameter list is mapped by DFSGVRMP. The storage for the parameter list is obtained by the calling application and must be GVRMP_LN in length. The address of the parameter list is specified either as a symbol or a register by using the PARMS parameter on the CALL request.

The CALL function returns an output area that contains the IMS installed level and that is mapped by DFSGVRMO. The address of the output area is in the GVRMP_A_OUTPUT_AREA field of the parameter list. Optionally, you can use the RETAREA parameter on the CALL request to specify a register that receives the address of the output area. Specifying the RETAREA parameter provides a way to prevent the calling application from having to reference the field in the parameter list directly.

If the RETAREA register or the parameter list field GVRMP_A_OUTPUT_AREA is nonzero after the output area is returned by the CALL function, regardless of the return code the calling application must use the REL function to release the output storage areas.

Syntax for CALL requests



Parameters for CALL requests

PARMS=*symbol* | (*R2-R12*)
Specifies the address of the parameter list to call module DFSGVRM0. The parameter list is mapped by DFSGVRMP.

If specified as a symbol, the symbol must represent an area of storage that is GVRMP_LN in length.

If specified as a register, the register must point to an area that is GVRMP_LN in length.

RETAREA=(*R2-R12*)
Specifies a register that receives the address of the output area that is returned by the DFSGVRM module and that is mapped by DFSGVRMO.

This parameter is optional.

If this parameter is not specified, the address of the output area is in the GVRMP_A_OUTPUT_AREA field of the parameter list.

The application that issues the CALL function is responsible for releasing the output area when the RETAREA register or the GVRMP_A_OUTPUT_AREA field is non-zero, regardless of the macro return code. The output area address is set to zero when the output area is not present on return.

LOC=24|31
Specifies whether the output area should be retrieved from below (**LOC=**24) or above (**LOC=**31) the 16 MB line.

This parameter is optional.

If this parameter is not specified, the area is above the 16 MB line (**LOC=**31).

Return and reason codes for CALL requests

When the DFSGVRM FUNC=CALL request is made, IMS returns the return code in register 15 and the reason code in register 0.

Table 110. Return and reason codes for CALL requests

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'00000008'	X'*yyyyzzzz*'	Unable to LOAD module DFSGVRM0. In the reason code: ***yyyy*** Is the low-order 2 bytes of the ABEND code that LOAD would have issued. ***zzzz*** Is the low-order 2 bytes of the reason code for the ABEND that would have been issued.
X'0000000C'	Return code from DELETE macro	Failure trying to delete module DFSGVRM0 after successful invocation of DFSGVRM0. The return area provided is valid and available for use.
X'00000010'	Return code from STORAGE RELEASE	Unable to free the output area that is referred to by the AREA= parameter.
X'00001008'	Return code from STORAGE OBTAIN	The DFSGVRM0 module could not obtain storage for the output area.
X'0000100C'	X'*yyyyzzzz*'	Unable to LOAD module DFSVC000. In the reason code: ***yyyy*** Is the low-order 2 bytes of the ABEND code that LOAD would have issued. ***zzzz*** Is the low-order 2 bytes of the reason code for the ABEND that would have been issued.
X'000010FF'	Various	An internal error occurred.

Related reference

“REL request (REL) for the IMS installed level API” on page 489
Use the REL function of the DFSGVRM API to release the storage that were returned by the CALL function for the output areas and that is no longer needed.

Chapter 46. REL request (REL) for the IMS installed level API

Use the REL function of the DFSGVRM API to release the storage that were returned by the CALL function for the output areas and that is no longer needed.

If the RETAREA register or the parameter list field GVRMP_A_OUTPUT_AREA is nonzero after the output area is returned by the CALL function, regardless of the return code the calling application must use the REL function to release the storage that were allocated for the output areas. The output area address is set to zero when the output area is not present on return.

Syntax for REL requests

Parameters for REL requests

AREA=(*R1-R12*)
Specifies a register that contains the address of the output area to be released.

Return and reason codes for REL requests

When the DFSGVRM FUNC=REL request is made, IMS returns the return code in register 15 and the reason code in register 0.

Table 111. Return and reason codes for REL requests

Return code	Reason code	Meaning
X'00000000'	X'00000000'	The request completed successfully.
X'00000010'	Return code from STORAGE RELEASE	Unable to free the output area that is referred to by the **AREA=** parameter.
X'000010FF'	Various	An internal error occurred.

Related reference

"CALL request (CALL) for the IMS installed level API" on page 485
Use the CALL function of the DFSGVRM API to obtain the version, release, and modification level of the IMS system.

Part 8. Repository Server batch interface (FRPBATCH)

The Repository Server (RS) address space batch interface (FRPBATCH) is a batch interface to manage the RS and the repositories.

Chapter 47. Commands for FRPBATCH

The Repository Server (RS) address space batch interface (FRPBATCH) is invoked from JCL as an executable job step program, and accepts commands through the SYSIN input stream.

Some **MODIFY (F)** and **FRPBATCH** commands are equivalent.

Table 112. Equivalent Modify (F) and FRPBATCH commands

MODIFY (F)	FRPBATCH	Note
--	ADD	
ADMIN DISPLAY	LIST	
ADMIN START	START	
ADMIN STOP	STOP	Stops the IMSRSC repository
--	RENAME	
--	DELETE	
ADMIN DSCHANGE	DSCHANGE	
--	UPDATE	
AUDIT	--	Changes the audit level
SECURITY	--	Refreshes in-storage profiles
SHUTDOWN	--	Stops the RS. Similar to the STOP command through the z/OS STOP (P) interface.

The job control statements are:

EXEC
Specifies the program name (PGM=FRPBATCH) and the program parameters. The parameters must be comma delimited and can be supplied in any order. Each parameter is in the format of `parameter=value`.

XCFGROUP
The name of the XCF group in which the RS is located. The value of this parameter is the same as the value of the XCF_GROUP_NAME parameter in the FRPCFG member of the IMS PROCLIB data set.

LANG
The language for output messages. Only ENU is supported. If this parameter is omitted, ENU is used.

The following example is an EXEC statement:

```
EXEC PGM=FRPBATCH,PARM=('XCFGROUP=FRPGRUP1','LANG=ENU')
```

SYSPRINT DD
Defines a data set for general messages and information. DCB attributes for this data set are RECFM=FBM and LRECL=133.

FRPLIST DD
Defines a data set for the LIST command output. If this statement is omitted, the output of the LIST command is written to the SYSPRINT DD output data set. DCB attributes for this data set are RECFM=FBM and LRECL=133.

The commands to be processed are specified on the SYSIN control cards. The SYSIN control cards must be entered in columns 1 - 72 of the input stream. Each statement has the following general form:

```
command_name parameter1 parameter2(value) /*inline comment
*Full-line comment
```

The following rules apply:

- The *command_name* is the first item of a command. It is followed by a space ( ), and one or more parameters.
- If a command contains more than one parameter, each parameter must be separated by one or more spaces ( ), a comma (,), or both.
- A parameter can have a value. The value of a parameter is enclosed in parentheses ().
- Command names, parameters, and values are converted by the program to uppercase.
- To enter an inline comment, type a forward slash followed by an asterisk (/*). Subsequent characters on the same line are ignored by the program.
- To enter a full-line comment, type an asterisk (*) in column 1. All characters on that line are ignored by the program.
- To break a command over multiple lines, use one of the following continuation characters:
  - Use a hyphen (-) to separate parameters for the same command across multiple lines. The hyphen does not delete the leading separator from continued lines. For example:

    ```
    RENAME REPOSITORY(REPOSITORY_NAME) -
    REPOSITORYNEW(REPNEWNAME)
    ```

  - Use a plus sign (+) to enter a single parameter and its value on multiple lines. It deletes the leading separator from continued lines. The plus sign must immediately follow the last character on a line. For example:

    ```
    START REPOSITORY
    (REPOSIT+
    ORY_NAME)
    ```

The following sample is JCL that runs the FRPBATCH interface with the various FRPBATCH commands:

```
//FRPBAT   EXEC PGM=FRPBATCH,PARM='XCFGROUP=FRPGRUP1'
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
//*
 ADD REPOSITORY(IMSRSC_REPOSITORY)+
     REPDSN1RID(IMSTESTS.REPO.IMSPRI.RID)+
     REPDSN1RMD(IMSTESTS.REPO.IMSPRI.RMD)+
     REPDSN2RID(IMSTESTS.REPO.IMSSEC.RID)+
     REPDSN2RMD(IMSTESTS.REPO.IMSSEC.RMD)+
     REPDSN3RID(IMSTESTS.REPO.IMSSPR.RID)+
     REPDSN3RMD(IMSTESTS.REPO.IMSSPR.RMD)+
     AUTOOPEN(YES)
 //*
 START REPOSITORY(IMSRSC_REPOSITORY)+
       MAXWAIT(30,CONTINUE)

 //*
 LIST REPOSITORY(IMSRSC_REPOSITORY)

 //*
 STOP REPOSITORY(IMSRSC_REPOSITORY)+
      MAXWAIT(30,CONTINUE)

 //*
 RENAME REPOSITORY(IMSRSC_REPOSITORY) REPOSITORYNEW(IMSRSC_TEST_REPOSITORY)

 //*
 UPDATE REPOSITORY (IMSRSC_TEST_REPOSITORY) -
 REPDSN1RID(IMSTESTS.TESTREPO.IMSPRI.RID) -
 REPDSN1RMD(IMSTESTS.TESTREPO.IMSPRI.RMD) -
 AUTOOPEN(NO)
```

```
//*
DELETE REPOSITORY(IMSRSC_TEST_REPOSITORY)
```

FRPBATCH commands provide the following functions:

Related concepts

Overview of the IMSRSC repository (System Definition)

IMSRSC repository administration (System Administration)

Starting and stopping the IMSRSC repository (Operations and Automation)

Opening the IMSRSC repository (Operations and Automation)

Related reference

F reposervername,ADMIN (Commands)

FRPCFG member of the IMS PROCLIB data set (System Definition)

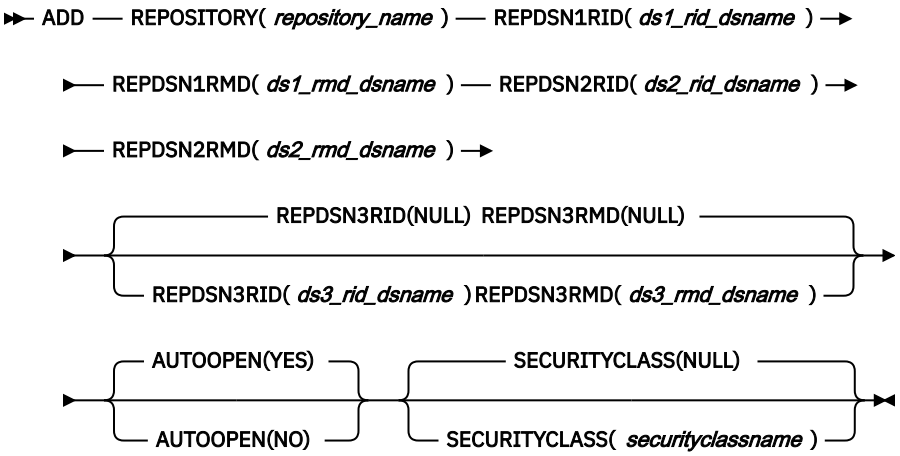
ADD command for FRPBATCH

Use the **ADD** FRPBATCH command to add an IMSRSC repository to the Repository Server (RS) catalog repository data sets.

Subsections:

- "Syntax" on page 495
- "Keywords" on page 495

Syntax



Keywords

REPOSITORY()

This is a required keyword. The name of the repository. The name can be up to 44 characters long. Valid characters are A-Z (uppercase only), 0-9, and the following symbols: number sign (#), dollar sign ($), at sign (@), period (.), and underscore (_). All lower case characters are converted to uppercase.

A repository name of CATALOG cannot be used, because it is reserved for RS usage.

REPDSN1RID()

This is a required keyword. The primary repository index data set (RID). A valid existing VSAM key sequenced data set (KSDS) name is required for the repository to initialize.

REPDSN1RMD()

This is a required keyword. The primary repository member data set (RMD). A valid existing VSAM KSDS name is required for the repository to initialize.

REPDSN2RID()
This is a required keyword. The secondary RID. A valid existing VSAM KSDS name is required for the repository to initialize.

REPDSN2RMD()
This is a required keyword. The secondary RMD. A valid existing VSAM KSDS name is required for the repository to initialize.

REPDSN3RID()
This keyword is optional. The spare RID. If this parameter is not specified, the spare data set is set to NONE. If specified, it must be a valid VSAM KSDS data set name or NULL. Setting the REPDSN3 data set pair to NULL results in its status being set to NONE. The spare is not allocated until data set recovery processing is initiated.

REPDSN3RMD()
This keyword is optional. The spare RMD. If this parameter is not specified, the spare data set is set to NONE. If specified, it must be a valid VSAM KSDS data set name or NULL. Setting the REPDSN3 data set pair to NULL results in its status being set to NONE. The spare is not allocated until data set recovery processing is initiated.

AUTOOPEN(YES | NO)
This keyword is optional. Specifies when repository data sets are allocated.

YES
Repository data sets are allocated when the repository is started. This is the default.

NO
Repository data sets are allocated when you first connect to the repository.

SECURITYCLASS(NULL | *securityclassname*)
This keyword is optional. Specifies the name of the security class to be used for the user repository. The name must be left-aligned, an 8-byte name with trailing contiguous spaces. The first character must be alphabetic and subsequent name characters alphanumeric.

If this parameter is omitted, or if NULL is specified, there is no security for this user repository. The name specified can be the same name as the value for SAF_CLASS specified for the RS address space in the FRPCFG member of the IMS PROCLIB data set.

Related concepts
Overview of the IMSRSC repository (System Definition)
IMSRSC repository and RS catalog repository data sets (System Definition)
Related tasks
Restricting access to the RS catalog repository and IMSRSC repository (System Administration)
Allocating the IMSRSC repository data sets (System Definition)
Related reference
FRPCFG member of the IMS PROCLIB data set (System Definition)

DELETE command for FRPBATCH

Use the **DELETE** FRPBATCH command to remove an IMSRSC repository from the Repository Server (RS) catalog repository data sets.

Note: When you remove a repository from the RS catalog repository, its data sets are not deleted. To delete the data sets, use the z/OS Access Method Services (IDCAMS) utility or a similar method after you have removed the repository from the RS catalog repository.

Subsections:

Syntax

Keywords

REPOSITORY()
This is a required keyword. The name of the repository to be removed. "CATALOG" is reserved for internal use and cannot be deleted.

Related concepts
Overview of the IMSRSC repository (System Definition)
Related tasks
Removing an IMSRSC repository from the RS catalog repository (System Administration)
Related reference
z/OS: DFSMS Access Method Services for Catalogs

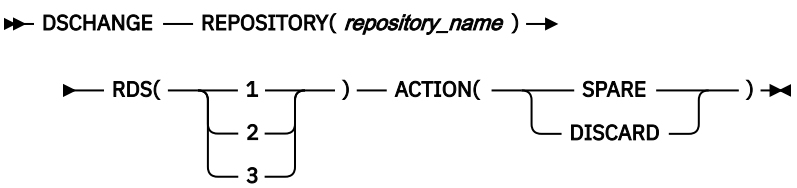
DSCHANGE command for FRPBATCH

Use the **DSCHANGE** FRPBATCH command to change the status of an IMSRSC repository data set pair to either DISCARD or SPARE.

Subsections:

- "Syntax" on page 497
- "Keywords" on page 497

Syntax



Keywords

REPOSITORY()
This is a required keyword. The name of the repository to be changed. "CATALOG" is reserved and cannot be changed.

RDS(1 | 2 | 3)
This is a required keyword. A number in the range of 1-3 to identify the repository data set pair to which the requested DSCHANGE action is to be applied.

1
The primary repository data set pair (COPY1).

2
The secondary repository data set pair (COPY2).

3
The spare repository data set pair (SPARE).

ACTION(SPARE | DISCARD)
This keyword is required. The action to be applied to the repository data sets that are specified in the RDS parameter.

SPARE
Request to change the repository data set pair disposition to SPARE status. The SPARE action can only be executed against a repository data set pair with DISCARD status. The SPARE repository index data sets (RIDs) and the SPARE repository member data sets (RMDs) must be empty.

DISCARD
Request to change the repository data set pair disposition to DISCARD status. The DISCARD action can be executed against either of the active repository data sets (the COPY1 or COPY2) or the SPARE repository data sets. A data set must be set to DISCARD status before a new data set can be defined. The repository must be in a stopped state for this request to process the DISCARD action against COPY1 or COPY2. The repository is not required to be stopped to process the DISCARD against the SPARE.

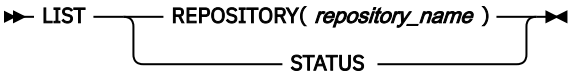
Related concepts
Overview of the IMSRSC repository (System Definition)
IMS repository data set states (System Definition)

LIST command for FRPBATCH

Use the **LIST** FRPBATCH command to list information about one IMSRSC repository or all repositories that are defined to the Repository Server (RS) catalog repository data sets.

Subsections:

Syntax



Keywords

REPOSITORY()
List details for a specific repository that is defined to the RS catalog repository data sets. The name of the repository for which to list information.

STATUS
List the details for all the repositories that are defined to the RS catalog repository data sets.

The following information is returned:

- Name of the repository
- Status of the repository
- Date when the repository was last updated and the user ID of the user who updated it

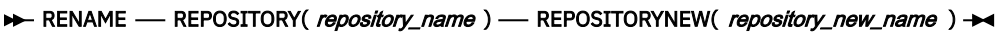
Related concepts
Overview of the IMSRSC repository (System Definition)
Related tasks
Viewing IMSRSC repository definitions and status (System Administration)

RENAME command for FRPBATCH

Use the **RENAME** FRPBATCH command to rename an IMSRSC repository.

Subsections:

Syntax



Keywords

REPOSITORY()
This is a required keyword. The current name of the repository to be renamed. "CATALOG" is reserved for internal use and cannot be renamed.

REPOSITORYNEW()
This is a required keyword. The new name of the repository to be renamed. The name can be up to 44 characters long. Valid characters are A-Z (uppercase only), 0-9, and the following symbols: period (.), underscore (_), number sign (#), dollar sign ($), and at sign (@). All lowercase characters are converted to uppercase.

A repository name of "CATALOG" cannot be used, because it is reserved for Repository Server (RS) usage.

Related concepts
Overview of the IMSRSC repository (System Definition)
Updating IMSRSC repository specifications in the RS catalog repository (System Administration)

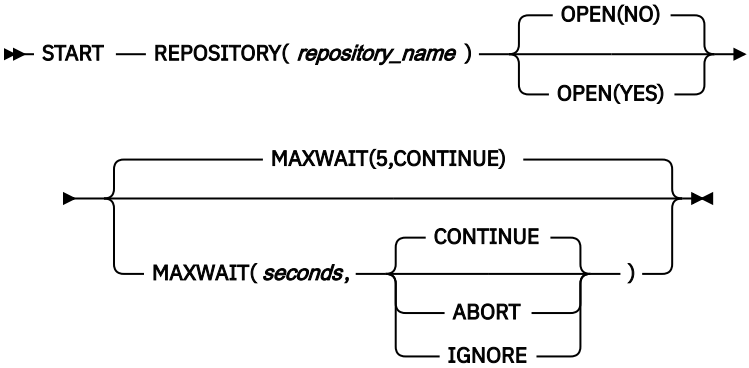
START command for FRPBATCH

Use the **START** FRPBATCH command to start an IMSRSC repository.

Subsections:

- "Syntax" on page 499
- "Keywords" on page 499

Syntax



Keywords

REPOSITORY()
This is a required keyword. The name of the repository data set to be started. The Repository Server (RS) catalog repository data sets cannot be started using the START command.

OPEN(NO | YES)
This keyword is optional. Specifies whether to open the repository data sets immediately after the repository is started.

NO
Repository data sets are opened when a user first connects to the repository, or immediately if AUTOOPEN=Y is defined for the repository with the **ADD** FRPBATCH command. This is the default.

YES
Repository data sets are allocated and opened when the repository is started.

MAXWAIT(5 | *seconds* , CONTINUE | IGNORE | ABORT)
This keyword is optional. The maximum number of seconds (0-9999) that the utility waits for the START operation to complete. The default is 5 seconds. A value of 0 means that the utility does not wait and returns a code immediately. MAXWAIT also specifies the action to take if the START operation is not complete when the MAXWAIT period expires (when the request to the server to start the user repository is successful but the batch utility is not able to confirm that the repository is in the requested state).

CONTINUE
Command processing continues even when the MAXWAIT period expires. The return code is set to 4.

IGNORE
Command processing continues even when the MAXWAIT period expires. No return code is set.

ABORT
Command processing is terminated when the MAXWAIT period expires. The return code is set to 8.

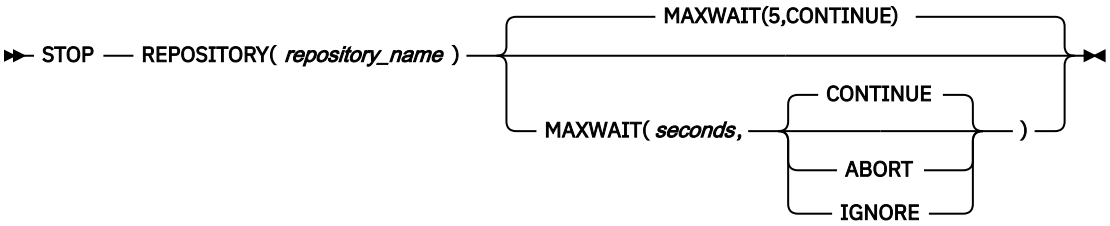
Related concepts
Overview of the IMSRSC repository (System Definition)

STOP command for FRPBATCH

Use the **STOP** FRPBATCH command to stop an IMSRSC repository that is defined to the Repository Server (RS) catalog repository data sets.

Subsections:

Syntax



Keywords

REPOSITORY()
This is a required keyword. The name of the repository data set to be stopped. The RS catalog repository data set cannot be stopped using the STOP command.

Requests that the RS stop this repository, preventing further connections to it. If the repository data sets are allocated, the server deallocates the data sets after all write operations are completed and closes the repository.

MAXWAIT(5 | *seconds* , CONTINUE | IGNORE | ABORT)
This keyword is optional. The maximum time (0-9999) in seconds that the utility waits for the STOP operation to be completed. The default is 5 seconds. A value of 0 means that the utility does not wait and returns a code immediately. MAXWAIT also specifies the action to be taken if the STOP operation has not been completed when the MAXWAIT period expires (when the request to the server to stop the user repository is successful but the batch utility is not able to confirm that the repository is in the requested state).

CONTINUE
Command processing continues even when the MAXWAIT period expires. The return code is set to 4.

IGNORE
Command processing continues even when the MAXWAIT period expires. No return code is set.

ABORT
Command processing is terminated when the MAXWAIT period expires. The return code is set to 8.

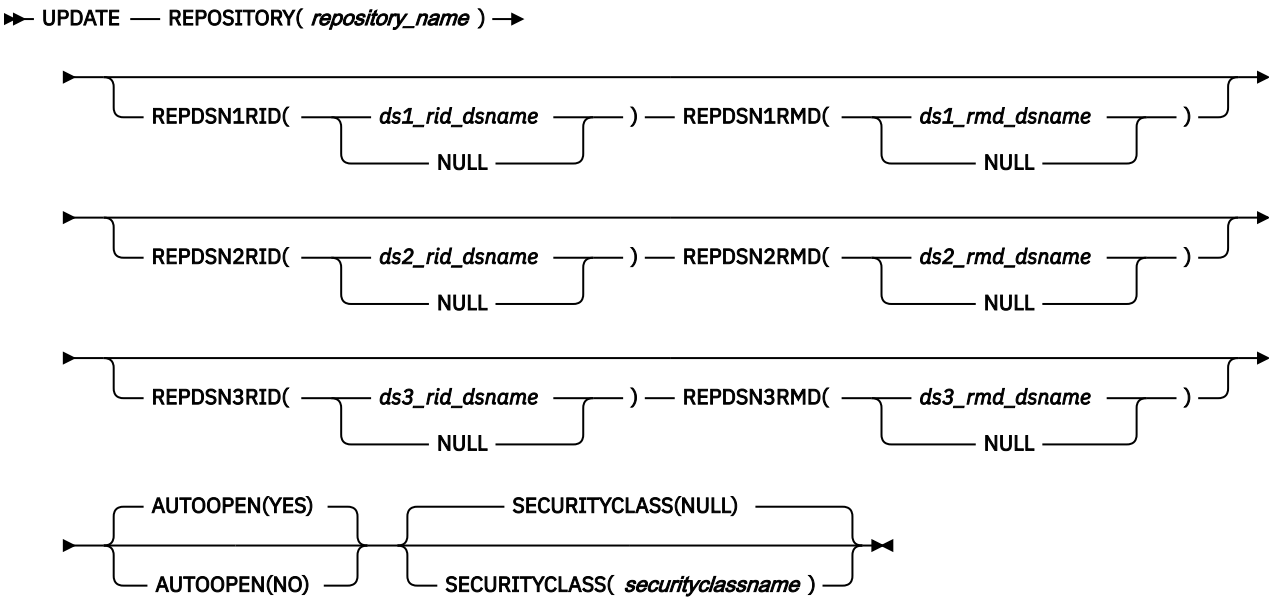
Related concepts
Overview of the IMSRSC repository (System Definition)

UPDATE command for FRPBATCH

Use the **UPDATE** FRPBATCH command to update an IMSRSC repository definition in the Repository Server (RS) catalog repository data sets. Use this command to change the IMSRSC repository data sets or the AUTOOPEN specification of a repository.

Subsections:

Syntax



Keywords

REPOSITORY()
This is a required keyword. The name of the repository to be updated. "CATALOG" is reserved and cannot be updated.

REPDSN1RID()
This is an optional keyword. The name of the primary repository index data set (RID). If this parameter is omitted, or if NULL is specified, the data set is not updated.

If you specify REPDSN1RMD, but do not want to update this parameter, you still must specify this parameter (with NULL).

REPDSN1RMD()
This is an optional keyword. The name of the primary repository member data set (RMD). If this parameter is omitted, or if NULL is specified, the data set is not updated.

If you specify REPDSN1RID(), but do not want to update this parameter, you still must specify this parameter (with NULL).

REPDSN2RID()
This is an optional keyword. The name of the secondary RID. If this parameter is omitted, or if NULL is specified, the data set is not updated.

If you specify REPDSN2RMD(), but do not want to update this parameter, you still must specify this parameter (with NULL).

REPDSN2RMD()
This is an optional keyword. The name of the secondary RMD. If this parameter is omitted, or if NULL is specified, the data set is not updated.

If you specify REPDSN2RID(), but do not want to update this parameter, you still must specify this parameter (with NULL).

REPDSN3RID()
This keyword is optional. The name of the spare RID. If this parameter is omitted, or if NULL is specified, the data set is not updated.

If you specify REPDSN3RMD(), but do not want to update this parameter, you still must specify this parameter (with NULL).

REPDSN3RMD()
This keyword is optional. The name of the spare RMD. If this parameter is omitted, or if NULL is specified, the data set is not updated.

If you specify REPDSN3RID(), but do not want to update this parameter, you still must specify this parameter (with NULL).

AUTOOPEN(YES | NO)
This keyword is optional. Specifies when repository data sets are allocated.

YES
Repository data sets are allocated when the repository is started. This is the default.

NO
Repository data sets are allocated when you first connect to the repository.

SECURITYCLASS(NULL | *securityclassname*)
This keyword is optional. Specifies the name of the security class to be used for the user repository. The name must be left-aligned, an 8-byte name with trailing contiguous spaces. The first character must be alphabetic and subsequent name characters alphanumeric.

If this parameter is omitted, or if NULL is specified, there is no security for this user repository. The name specified can be the same name as the SECURITYCLASS specified for the RS address space in the FRPCFG member of the IMS PROCLIB data set.

Related concepts
Overview of the IMSRSC repository (System Definition)
Updating IMSRSC repository specifications in the RS catalog repository (System Administration)
Related reference
FRPCFG member of the IMS PROCLIB data set (System Definition)

Part 9. VTAM and SNA reference information

The topics in this section provide reference information about VTAM and SNA.

Chapter 48. Bind parameters for SLU P and LU 6.1

You can search these topics for the session parameters that IMS specifies when establishing connection using the VTAM OPNDST macro instruction. These parameters define the rules that a logical unit must follow when communicating with IMS.

Restriction: A VTAM restriction exists on the OUTBUF and RECANY buffer sizes for logical units requiring a bind from IMS.

The following outbound (from IMS to the SLU) BIND formats apply to the device types indicated in the following topics.

Related reference

“Format for CINIT user data parameters” on page 525
This topic describes the CINIT user data parameters and the syntax rules for them.

Finance communication system bind parameters

You can use this table to search for the session parameters that are defined as UNITYPE=FINANCE or UNITYPE=SLUTYPEP on the TYPE macro or an Extended Terminal Option (ETO) logon descriptor for a Finance Communication System.

Table 113. Finance communication system bind parameters

Byte	Bit	Content	BIND FORMAT description
0		X'31'	
1		X'01'	Format=0; BINDTYPE=COLD[1]
2		X'04'	FM Profile 4
3		X'04'	TS Profile 4
4			Primary NAU protocol
	0	B'1'	Multiple RU chains
	1	B'0'	Immediate request
	2-3	B'11'	Any response
	4-5	B'00'	Reserved
	6	B'0'	No compression
	7	B'1'	Primary can send EB
5		B'1'	Secondary NAU protocol
	0	B'1'	Multiple RU chains
	1	B'0'	Immediate request
	2-3	B'11'	Any response
	4-5	B'00'	Reserved
	6	B'0'	No compression
	7	B'1'	Secondary can send EB

Table 113. Finance communication system bind parameters (continued)

Byte	Bit	Content	BIND FORMAT description
6		X'60'	Common NAU protocol
	0	B'0'	Reserved
	1	B'1'	Message headers are allowed
	2	B'1'	Brackets to be used
	3	B'0'	Bracket termination rule 2 (unconditional)
	4	B'0'	No alternate code
	5-7	B'000'	Reserved
7		X'80'	Common NAU protocol
	0-1	B'10'	FM transmission mode - HDX flip/flop
	2	B'0'	Primary ERP responsibility
	3	B'0'	Secondary station is the first speaker
	4-5	B'00'	Reserved
	6	B'0'	No related chains
	7	B'0'	Contention resolution
8			Reserved: X'00' should be specified
9			SLU receive pacing count: Not changed by IMS[2]
10			SLU max. RU send size: Set to maximum receive any buffer size from IMS system definition[1, 3, 4]
11			PLU max. RU send size: Set from output buffer size specified on IMS system definition[1, 4]
12-13			Reserved: X'0000' should be specified
14			LU TYPE: X'00' should be specified
15-26			Reserved: XL11'00' should be specified
27			PLU name length[2]
28-35			PLU name: IMS ACB name[2]
36		X'0B'	User data length
37-47			User data[5]
37		X'00'	Structured data indicator
38		X'09'	Length of USERVAR segment
39		X'03'	USERVAR segment indicator
40-47		XRF USERVAR	USERVAR

Table 113. Finance communication system bind parameters (continued)

Byte	Bit	Content	BIND FORMAT description

Notes:

1. Bytes 1 through 7, 10, and 11 are set to the indicated values set by IMS and cannot be changed by the user. The pacing parameter is defined in the VTAM list LU definition or through the mode table entry for the LU (VTAM DLOGMOD parameter).
2. Bytes 0, 9, and 27 through 35 are set by VTAM.
3. The receive-any buffer size is determined by the user-supplied value for size on the RECANY keyword parameter of the COMM macro statement, less 28 bytes.
4. IMS does not set the buffer size for the 4701/4702.
5. When the BIND data issued is an XRF system that uses the USERVAR instead of MNPS, the following structured user segment is included:

   Length of USERVAR segment - X'09'
   USERVAR segment indicator - X'03'
   USERVAR (8 bytes) - USERVAR of the XRF system

Related reference

z/OS: Request unit (RU) formats

IMS as primary half session

A specific bind format is sent by IMS during an IMS-to-other session initiation. If the mode table entry indicates negotiated bind, IMS overrides the mode table primary NAU protocol field with the indicated values prior to sending the bind.

IMS allows some parameters to be optionally set by a VTAM mode table entry or negotiated bind response. IMS then operates within the indicated constraints. For a non-negotiated bind, IMS checks the parameters for validity before sending the bind. For negotiated bind, IMS checks the parameters for validity prior to sending the BIND request and upon receipt of the bind response, because the secondary half session can modify parameters within the constraints indicated in the following table.

Table 114. Logical unit type 6.1 bind parameters

Byte	Bit	Content	BIND FORMAT description[1]
0		X'31'	Bind request code
1	0-3	B'0000'	FORMAT: TYPE 0
	4-7	B'0000', B'0001'	BIND TYPE: 0000 - negotiated 0001 - non-negotiated
2	0-7	X'12'	FM Profile 18
3	0-7	X'04'	TS Profile 4

Table 114. Logical unit type 6.1 bind parameters (continued)

Byte	Bit	Content	BIND FORMAT description[1]
4			Primary NAU protocol
	0	B'1'	Multiple RU chains
	1	B'0'	Immediate request
	2-3	B'10', B'11' (B'11' set for negotiated BIND)	10-Definite response chains 11-Exception/Definite response chains
	4	B'0'	Cannot send prepare
	5	B'0'	Reserved
	6	B'0'	No compression
	7	B'1'	0-Primary cannot end bracket 1-Primary can send end bracket
5			Secondary NAU protocol
	0	B'1'	0-Single RU chains, 1-Multiple RU chains
	1	B'0'	Immediate request
			01-Exception response chains
			10-Definite response chains
	2-3	B'10', B'11'	11-Exception/Definite response chains
	4	B'0'	Cannot send prepare
	5	B'0'	Reserved
	6	B'0'	No compression
			0-Secondary cannot end bracket
	7	B'0', B'1'	1-Secondary can send end bracket

Table 114. Logical unit type 6.1 bind parameters (continued)

Byte	Bit	Content	BIND FORMAT description[1]
6			Common NAU protocol
	0	B'0'	Reserved
	1	B'1'	Function management headers allowed
			Bracket state
			0 - Bracket state mgr reset to in-bracket state
	2	B'0',B'1'	1 - Bracket state mgr reset to between-bracket state[2]
	3	B'1'	Conditional bracket termination
	4	B'0'	No alternate code
	5	B'0' Sequence numbers not available B'1' Sequence numbers available	STSN request flag
	6	B'0' BIS not sent B'1' BIS sent	BIS sent flag
	7	B'0'	Reserved
7	0-1	B'10'	FM transaction mode, HDX-FF
	2	B'1'	Sender ERP
	3	B'0'	Secondary is first speaker
	4-6	B'0000'	Reserved
			If byte 6 bit 2 is 0, then
			0-Secondary speaks first after data traffic active state
	7	B'0', B'1'	1-Primary speaks first after data traffic active state[2]
8	0-7	Unchanged	Secondary send pacing count
9	0-7	Unchanged	Secondary receive pacing count
10	0-7	Set from user Define receive-any	SLU max send RU size[3]
11	0-7	Set from user Define outbuffer size	PLU max send RU size
12	0-7	Unchanged	Primary send pacing count
13	0-7	Unchanged	Primary receive pacing count
			Presentation Services
14	0-7	X'06'	LU profile (LUTYPE6)
15	0-7	X'00' Reserved	Function management header subset

Table 114. Logical unit type 6.1 bind parameters (continued)

Byte	Bit	Content	BIND FORMAT description[1]
16	0	Reserved	Primary half-session flags
	1	Reserved	
	2	1 - FMH6: Receive SYSMSG supported	
	3	1 - Receive SCHEDULER model supported	
	4	1 - Receive QMODEL supported	
	5	0 - Linear file model ignored	
	6	0 - DL/I model ignored	
	7	Reserved	
17		Reserved	
18-19		Reserved	
20	0	Reserved	Secondary half-session flags
	1	Reserved	
	2	1 - Receive SYSMSG supported	
	3	0 - Receive SCHEDULER not supported[4] 1 - Receive SCHEDULER model supported	
	4	0 - Receive QMODEL ignored 1 - Receive QMODEL supported	
	5	0 - linear file model ignored	
	6	0 - DL/I model ignored	
	7	Reserved	
21		Reserved	
22-26		Reserved	
27	0-7		Length of PLU name
28-M			PLU name
M+1	0-7	X'00' No user data present	Length of user data
M+2-N		X'00' Structured fields follow X'00' First byte of unstructured user data[5]	User data

Table 114. Logical unit type 6.1 bind parameters (continued)

Byte	Bit	Content	BIND FORMAT description[1]
M+3-N		Remainder of unstructured user data	For unstructured user data
M+3-N		Structured fields	For structured user data[6]
N+1	0-7	Structured fields, request correlation (URC) field X'00' = no URC present	Length of URC[7]
N+2-P		End-user-defined identifier	URC[4]
P+1	0-7	X'00'=no secondary name	Length of secondary LU name[4]
P+2-R	0-7	Secondary LU name	Secondary LU name[4]

Notes:

1. The length of the BIND RU cannot exceed 256 bytes; otherwise, a negative response is returned.
2. Set to indicate possible in-bracket or process restart. Set by IMS on bind when response mode output remains on the queue or when IMS is in conversational mode. Can also be sent on a negotiated bind response by the other half session.
3. The receive-any buffer size is determined by the user-supplied value for size on the RECANY keyword parameter of the COMM macro statement, less 28 bytes.
4. When the bind indicates that the other half session does not support the SCHEDULER process, IMS sends all unsolicited and asynchronous output using ATTACH.
5. Unstructured user data is ignored and not provided by IMS.
6. Structured user data formats. A structured field contains architected or subsystem-defined information and provides a means for subsystems to communicate data. Each structured field contains a field identifier (subfield number) and length. A structured data field can contain unstructured data.

   If structured field (M+3-N) is X'00', it contains unstructured data as follows:

   1
   Length of unstructured data field (including subfield key field). If zero, this field can be omitted entirely.

   2
   Subfield key: X'00'

   3-N
   Unstructured data. If the structured subfield number is X'03', an 8-byte USERVAR name follows the subfield.

   If the structured field (M+3-N) is X'01', it contains a session qualifier as follows:

   1
   Length of session qualifier field (including subfield key field). If zero, this field can be omitted entirely.

   2
   Subfield key: X'01'

   3
   Length of primary resource qualifier (X'00' means no primary source qualifier is present). Values 0 to 8 are valid.

   4-N
   Primary resource qualifier

N+1
Length of secondary resource qualifier (X'00' means no secondary resource qualifier is present). Values 0 to 8 are valid.

N+2-M
Secondary resource qualifier

M+1
Length of password (X'00' means no password is present). Values 0 to 8 are valid.

M+2-P
Password. Ignored on bind or bind reply from IMS and is not sent on bind or bind reply. IMS indexes structured fields to find field X'01', a session-qualifier field, when these parameters are required for session initiation using parallel sessions.

M+3-N
If the structured field has a subfield of X'02', IMS interprets the field as an MSC partner ID. If bind is issued in an XRF environment that uses USERVAR instead of MNPS, an additional structured segment is included in the user data. The format of this segment is:

1
Length of USERVAR segment, X'09'

2
Subfield key, X'03'

3-10
8-byte field containing USERVAR of the XRF complex

7. The URC and secondary LU name are not used by IMS but are shown for compatibility purposes.

Related reference
z/OS: Request unit (RU) formats

IMS as secondary half session

A specific bind format can be received by IMS during an IMS-to-other session initiation. If the bind parameters that are received indicate a negotiated BIND request, IMS overrides the secondary NAU protocol field with the indicated values before sending the bind response.

IMS allows some parameters to be optionally set using the bind and operates within the indicated constraints.

Table 115. IMS-to-other secondary half session

Byte	Bit	Content	BIND FORMAT description [1]
0		X'31'	Bind Request Code
1	0-3	B'0000'	Format: TYPE 0
	4-7	B'0000', B'0001'	Bind type: 0000 - negotiated 0001 - non-negotiated
2	0-7	X'12'	FM profile 18
3	0-7	X'04'	TS profile 4

Table 115. IMS-to-other secondary half session (continued)

Byte	Bit	Content	BIND FORMAT description [1]
4			Primary NAU Protocol
	0	B'0', B'1'	**B'0'** Single RU chains **B'1'** Multiple RU chains
	1	B'01', B'10'	Immediate request: **B'01'** Exception response chains **B'10'** Definite response chains
	2-3	B'01', B'10', B'11'	11-Exception/Definite response chains
	4	B'0'	Cannot send prepare
	5	B'0'	Reserved
	6	B'0'	No compression
	7	B'0', B'1'	0-Primary cannot end bracket 1-Primary can send end bracket
5			Secondary NAU Protocol
	0	B'1'	Multiple RU chains
	1	B'0'	Immediate request
	2-3	B'10', B'11' (B'11' set for negotiated bind)	10-Definite response chains 11-Exception/Definite response chains
	4	B'0'	Cannot send prepare
	5	B'0'	Reserved
	6	B'0'	No compression
	7	B'1'	0-Secondary cannot send end bracket 1-Secondary can send end bracket
6			Common NAU Protocol
	0	B'0'	Reserved
	1	B'1'	Function Management headers allowed
	2	B'0', B'1'	Bracket state: **B'0'** Bracket state manager reset to in-bracket state **B'1'** Bracket state manager reset to between-bracket state[2]

Table 115. IMS-to-other secondary half session (continued)

Byte	Bit	Content	BIND FORMAT description [1]
	3	B'1'	Conditional bracket termination
	4	B'0'	No alternate code
	5	B'0', B'1'	STSN required flag: **B'0'** Sequence numbers not available **B'1'** Sequence numbers available
	6	B'0', B'1'	BIS sent flag: **B'0'** BIS not sent **B'1'** BIS sent
	7	B'000'	Reserved
7	0-1	B'10'	FM transaction mode, HDX-FF
	2	B'1'	Sender ERP
	3-6	B'0000'	Reserved
	7	B'0', B'1'	IF byte 6 bit 2 is 0: 0 - Secondary speaks first after data traffic active state 1 - Primary speaks first after data traffic active state[2]
8	0-7	Unchanged	Secondary send pacing count
9	0-7	Unchanged	Secondary receive pacing count
10	0-7	Must ≥ defined outbuffer size	SLU max send RU size
11	0-7	Must be ≤ define receive-any size	PLU max send RU size[3]
12	0-7	Unchanged	Primary send pacing count
13	0-7	Unchanged	Primary receive pacing count
14	0-7	X'06'	LU profile (LUTYPE6)
15	0-7	X'00' Reserved	Function Management header subset

Table 115. IMS-to-other secondary half session (continued)

Byte	Bit	Content	BIND FORMAT description [1]
16	0	Reserved	Primary half-session flags
	1	Reserved	
	2	1 - Receive SYSMSG supported	
	3	0 - Receive SCHEDULER model not supported[4]	
		1 - Receive SCHEDULER model supported	
	4	0 - Receive QMODEL not supported	
		1 - Receive QMODEL supported	
	5	0 - linear file model not supported	
	6	0 - DL/I model not supported	
	7	Reserved	
17		Reserved	
18-19		Reserved	
20	0	Reserved	Secondary half-session flags
	1	Reserved	
	2	1 - FMH6: Receive SYSMSG supported	
	3	1 - Receive schedule model supported	
	4	1 - Receive QMODEL supported	
	5	0 - linear file model not supported	
	6	0 - DL/I model not supported	
	7	Reserved	
21		Reserved	
22-26		Reserved	
27	0-7		Length of PLU name

Table 115. IMS-to-other secondary half session (continued)

Byte	Bit	Content	BIND FORMAT description [1]
28-M			PLU name
M+1	0-7	X'00' No user data present	Length of user data
M+2-N		X'00' Structured fields follow X'00' first byte of unstructured user data[5]	User data
M+3-N		Remainder of unstructured user data	For unstructured user data
M+3-N		Structured fields[6]	For structured user data
N+1	0-7	Length of user request correlation (URC) field X'00' = no URC present	Length of URC[7]
N+2-P		URC: end-user defined identifier	URC[7]
P+1	0-7	X'00'=no secondary name	Length of secondary LU name[7]
P+2-R	0-7	Secondary LU name	Secondary LU name[7]

Notes:

1. The length of the BIND RU cannot exceed 256 bytes; otherwise, a negative response is returned.
2. Set to indicate possible in-bracket or process restart. Set by IMS on bind when response mode output remains on the queue or when IMS is in conversational mode. Can also be sent on a negotiated bind response by the other half session.
3. The receive-any buffer size is determined by the user-supplied value for size on the RECANY keyword parameter of the COMM macro statement, less 28 bytes.
4. When the bind indicates that the other half session does not support the SCHEDULER process, IMS sends all unsolicited or asynchronous output using ATTACH.
5. Unstructured user data is ignored and not provided by IMS.
6. Structured user data formats. A structured field contains architected or subsystem-defined information and provides a means for subsystems to communicate data. Each structured field contains a field identifier (subfield number) and length. A structured data field can contain unstructured data.

   If structured field (M+3-N) is X'00', it contains unstructured data as follows:

   1
   Length of unstructured data field (including subfield key field). If zero, this field can be omitted entirely.

   2
   Subfield key: X'00'

   3-N
   Unstructured data If the structured subfield number is X'03', an 8-byte USERVAR name follows the subfield.

   If the structured field (M+3-N) is X'01', it contains a session qualifier as follows:

   1
   Length of session qualifier field (including subfield key field). If zero, this field can be omitted entirely

   2
   Subfield key: X'01'

3
Length of primary resource qualifier (X'00' means no primary source qualifier is present). Values 0 to 8 are valid.

4-N
Primary resource qualifier

N+1
Length of secondary resource qualifier (X'00' means no secondary resource qualifier is present). Values 0 to 8 are valid.

N+2-M
Secondary resource qualifier

M+1
Length of password (X'00' means no password is present). Values 0 to 8 are valid.

M+2-P
Password (ignored on bind or bind reply received by IMS and not sent on bind or bind reply. IMS indexes structured fields to find field X'01', a session-qualifier field, when these parameters are required for session initiation using parallel sessions.

When the BIND data issued is an XRF system that uses USERVAR instead of MNPS, the following structured user segment is included:

Length of USERVAR segment - X'09'
USERVAR segment indicator - X'03'
USERVAR (8 bytes) - USERVAR of the XRF system

7. The URC and secondary LU name are not used by IMS but are shown for compatibility purposes.

Related reference
z/OS: Request unit (RU) formats

Chapter 49. Bind parameters for SLU 1 and SLU 2

IMS specifies different session parameters when establishing connection with SLU 1 and SLU 2. These parameters define the rules that a logical unit must follow when communicating with IMS.

The following outbound (from IMS to the SLU) BIND formats apply to the device types indicated in this section.

Related reference

“Format for CINIT user data parameters” on page 525
This topic describes the CINIT user data parameters and the syntax rules for them.

SLU 1 bind parameters

You can use this table to search for SLU 1 bind parameters.

Table 116. SLU 1 bind parameters

Byte	Bit	Content	Description
0		X'01'	Format=0; BINDTYPE=COLD
1		X'03'	FM Profile 3
2		X'03'	TS Profile 3
3			**PRIMARY LU PROTOCOLS**
	0	B'1'	Multiple RU chains
	1	B'0'	Immediate Request Mode
	2-3	B'11'	Chain Response Protocol: Any
	4-5	B'00'	Reserved
	6	B'0'	No Compression
	7	B'1'	Primary can send End Bracket
4			**SECONDARY LU PROTOCOLS**
	0	B'1'	Multiple RU Chains
	1	B'0'	Immediate Request Mode
	2-3	Note[1]	Chain Response Protocol
	4-5	B'00'	Reserved
	6	B'0'	No Compression
	7	B'0'	Secondary does not send End Bracket

Table 116. SLU 1 bind parameters (continued)

Byte	Bit	Content	Description
5			**COMMON LU PROTOCOLS (FIRST BYTE)**
	0	B'0'	Reserved
	1	Note[2]	FM Headers
	2	B'1'	Brackets can be used
	3	B'1'	Bracket Termination Rule 1 (Conditional)
	4	B'0'	EBCDIC (No Alternate Code)
	5-7	B'000'	Reserved
6			**COMMON LU PROTOCOLS (SECOND BYTE)**
	0-1	B'10'	Half-Duplex Flip-Flop
	2	B'0'	Primary ERP Responsibility
	3	B'0'	Secondary is First Speaker
	4-6	B'000'	Reserved
	7	B'0'	Secondary is Contention Winner
7	0-1	B'00'	Reserved
	2-7		SLU Send Pacing Count (Set by VTAM)[3]
8	0-1	B'00'	Reserved
	2-7		SLU Receive Pacing Count (Set by VTAM)[3]
9	0-7	Set from user-defined receive any	SLU to PLU RU Size
10	0-7	Set from user-defined outbuf size	PLU to SLU RU Size
11	0-1	B'00'	Reserved
	2-7		PLU CPMGR Send Pacing Count (Set by VTAM)[3]
12	0-1	B'00'	Reserved
	2-7		PLU CPMGR Receive Pacing Count (Set by VTAM)[3]
13	0-7	X'01'	LU Profile (LUTYPE1)
14-35		See Notes	Remainder of Bind Area

Table 116. SLU 1 bind parameters (continued)

Byte	Bit	Content	Description

Notes:

1. IMS forces flip-flop mode.
2. The preceding bind (bytes 0-6 and 13) overrides anything that might be in a logmode entry.
3. You can do this by coding the VPACING parameter on the VTAM list LU definition or by specifying the appropriate mode table entry on the LU definition by using the VTAM DLOGMOD parameter.
4. The remainder of the bind (bytes 14-35) can be specified if it is required by the device. It is taken from the logmode entry.
5. Unattended operation must be specified in the logmode entry using the following:

   BYTE 18

   Bit 0

   0
   Initiates attended

   1
   Initiates unattended

   Bit 1

   0
   Does not alternate from attended/unattended during session

   1
   Alternates from attended/unattended during session

   IMS forces attended mode if the node is defined as the master terminal.
6. IMS users should make the logmode entry according to the IMS definition.
7. If the terminal sends SCS2 transparent fields (identified by a X'35' followed by one byte containing the field's length, followed by the transparent field), the bind image bit BINPDSB1=BINTRNDS (offset 17=01) must be set. IMS processes these fields by deleting the X'35' and length byte, and by passing the unaltered transparent field to the editing routine.
8. IMS accepts a setting of B'10' (definite response), B'01' (exception response), or B'11' (either response). If a setting of B'00' is found, IMS sets B'01' if only the first component is defined, and B'10' if more than one component is defined.
9. IMS leaves this bit on (FM headers allowed). If off, IMS leaves it off (FM headers not allowed) if only one component is defined, and sets it on if more than one component are defined.

SLU 2 bind parameters

You can use this table to search for the bind parameters for SLU 2 devices.

Table 117. SLU 2 bind parameters

Byte	Bit	Content	Description
0		X'01'	Format=0; BINDTYPE=COLD
1		X'03'	FM Profile 3
2		X'03'	TS Profile 3

Table 117. SLU 2 bind parameters (continued)

Byte	Bit	Content	Description
3			Primary NAU protocol
	0	B'1'	Multiple RU chains
	1	B'0'	Immediate request
	2-3	B'11'	Any response
	4-5		Reserved
	6	B'0'	No compression
	7	B'1'	Primary can send EB
4			Secondary NAU protocol
	0	B'1'	Multiple RU chains
	1	B'0'	Immediate request
	2-3	B'01'	Exception response
	4-5		Reserved
	6	B'0'	No compression
	7	B'0'	Secondary cannot send EB
5			Common NAU protocol
	0		Reserved
	1	B'0'	Message headers are not allowed
	2	B'1'	Brackets to be used
	3	B'1'	Bracket termination rule 1 (conditional)
	4	B'0'	No alternate code
	5-7		Reserved
6			Common NAU protocol
	0-1	B'10'	FM transmission mode: HDX flip/flop
	2	B'0'	Primary ERP responsibility
	3	B'0'	Secondary station is the first speaker
	4-6	xxx	Reserved
	7	B'0'	For HDX-FF mode, secondary sends first when leaving data traffic reset state
7			SLU Send Pacing count: Not changed by IMS
8			SLU Receive Pacing count: Not changed by IMS
9			SLU MAX RU send size: Set to Max receive any buffer size from IMS system definition
10			PLU MAX RU send size: Set from output buffer size specified at IMS system definition
11-12			Reserved: X'0000' should be specified

Table 117. SLU 2 bind parameters (continued)

Byte	Bit	Content	Description
13			LU TYPE: Set to X'02' by IMS
14-18			Reserved: X'00' should be specified
19-20			User specified screen size if 3274/3276 device. Otherwise not changed.
21-22			Alternate screen size, X'00' should be specified.
23			Set to X'7E' if 3274/3276 NDS device or X'02' if non-NDS 3270 master terminal. Otherwise should be X'00'.
24-25			User specified: should be X'00'
26			PLU Name length
27-34			PLU name: IMS ACB name
35			User data length: Not supported, X'00' must be specified

Notes:

1. Bytes 0-6, 9, and 10 are set to the indicated values by IMS and cannot be changed by the user.
2. Byte 8 and bytes 26 through 34 are set by VTAM. You should set the remaining bytes to 0 (zero), but it is not mandatory.

Chapter 50. Format for CINIT user data parameters

This topic describes the CINIT user data parameters and the syntax rules for them.

For all non-MSC VTAM terminal types, IMS can receive user data parameters from the following sources:

- IMS (**/OPNDST** command)
- IMS autologon request
- User logon (such as SNA **INITSELF** command)
- Installation Logon exit routine (DFSLGNX0)
- Signon exit routine (DFSSGNX0)
- Destination Creation exit routine (DFSINSX0)

User data parameters are optional for migration purposes. However, when a match must be made between the terminal and user for session initiation, user data parameters are required for:

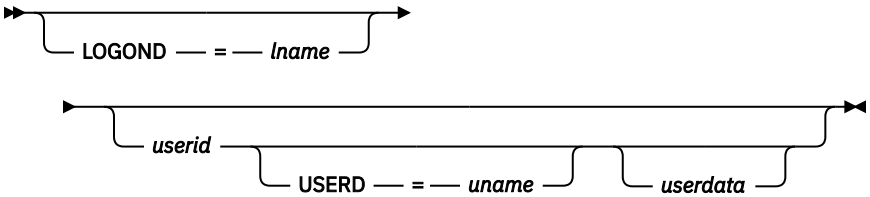
- ISC parallel session (LU 6.1 architecture)
- Finance (3601) and SLU P terminals, when used with ETO

All parameters, optional and required, appear in the CINIT user data field and are available to IMS when the VTAM Logon exit routine is scheduled.

During logon and signon processing, IMS performs minimal processing on CINIT user data parameters before calling the Installation Logon exit routine (DFSLGNX0) and the Signon exit routine (DFSSGNX0). If the Logon or Signon exit routines are not supplied, IMS assumes a default user data format.

User data format

The format for the CINIT user data is shown in the following syntax diagram.



For these parameters, blanks are required and are the only recognized delimiters. Do not use more than one blank to delimit parameters.

Parameters

LOGOND=*lname*
Specifies logon descriptor name to be used for the terminal attempting to log on to IMS. *lname* is one to eight bytes in length. The LOGOND and USERD parameters are valid only for Extended Terminal Operation (ETO).

userid
Specifies the 1- to 8-byte user ID of the user logging on to IMS. The *userid* parameter indicates that the user associated with this ID will also sign on to IMS.

The *userid* parameter is required for Finance, SLU P, ISC, and output-only devices (such as printers).

Restriction: ISC is restricted to only the user ID that translates to the ISC SUBPOOL name (SNA PHS/SHS qualifier). For ISC parallel sessions, the DFSLGNX0 exit routine receives only the user ID that translates to the ISC subpool name (SNA PHA/SHA qualifier).

USERD=*uname*
Specifies the 1- to 8-byte user descriptor name to be used to create the user control block structure at signon.

userdata
Specifies additional data for the Signon exit routine (DFSSGNX0) or the Sign On/Off Security exit routine (DFSCSGN0) and security products, such as RACF. For the exit routines, your installation defines the format of the *userdata* fields.

For RACF, the format of the *userdata* is:

userpw
Identifies the 1- to 8-byte user password that is associated with the previously entered *userid*. No keyword precedes the user password.

GROUP *groupname*
Identifies groupname as the 1- to 8-byte group name that is associated with the *userid* parameter. The GROUP keyword and associated parameter are optional.

NEWPW *nuserpw*
Identifies *nuserpw* as a new user password that replaces the current password, *userpw*.

The ETO logon descriptor name, *lname*, applies to IMS logon processing. All remaining optional parameters, however supplied, are passed to ETO user allocation, signon processing, and the security product, such as RACF. If a security product is used, all parameters not applicable to that product must be deleted before it is called. The parameters can be deleted in the DFSLGNX0, DFSSGNX0, and DFSCSGN0 exit routines.

Related concepts

z/OS Communications Server SNA Programmer's LU 6.2 Guide

Related reference

“Bind parameters for SLU P and LU 6.1” on page 505
You can search these topics for the session parameters that IMS specifies when establishing connection using the VTAM OPNDST macro instruction. These parameters define the rules that a logical unit must follow when communicating with IMS.

“Bind parameters for SLU 1 and SLU 2” on page 519
IMS specifies different session parameters when establishing connection with SLU 1 and SLU 2. These parameters define the rules that a logical unit must follow when communicating with IMS.

Chapter 51. SNA character string controls

SNA character string controls (SCS) describe specific functions for the EBCDIC control codes. You can use the primary functions that are described to format a printed page or an alphanumeric display screen.

Functions are also defined for codes that set modes of device operation, define data to be used in a unique fashion, or are used for communication between a device operator and an application program (where the specific function associated with the code is defined in a protocol established between a program and an operator).

An SCS data stream consists of a sequential string of control and data characters. Control function characters in the form of SCS-defined control codes can be intermixed with graphic data characters. Other data types (such as binary and packed decimal) are permitted only in conjunction with the transparent (TRN) control.

SCS control codes appear within the data portion of the request unit (RU). A function management (FM) header can precede SCS data within an RU. Functions such as component selection are performed by FM header functions and are not included as SCS functions.

SCS functions do not include data flow control functions, even when these functions are available to a keyboard operator through keys on the keyboard. For example, CANCEL is a data flow control request that can be initiated by a key on the keyboard.

Format controls

Formatting control functions format the output media at the device on a line and page basis. In addition to these controls, a device using SCS also automatically formats the character string to fit the line length of the device.

Automatic new line generation eliminates device line length dependencies from those applications in which a specific output format is not required. Therefore, the same character string is sent to devices with varying line length capabilities without the requirement to reformat the character string.

Where specific line and page formats are required, the formatting control functions are used. The automatic new line feature is always active; however, a character string formatted for a given line length can be presented on a device with a shorter line length without loss of data, but the format is lost. When the situation is reversed, where a character string constructed for a maximum presentation position is less than the line length of the device to which it is directed, use of the smaller maximum presentation position allows the string to be presented without loss of format.

Control function code assignments

You can search this table for SCS control functions that are assigned to EBCDIC codes.

Table 118. Control function code assignments

EBCDIC code	Function	Function abbreviation
04	Vertical Channel Select	VCS[1]
05	Horizontal Tab	HT
0B	Vertical Tab	VT
0C	Form Feed	FF
0D	Carrier Return	CR
14	Enable Presentation	ENP

Table 118. Control function code assignments (continued)

EBCDIC code	Function	Function abbreviation
15	New Line	NL
16	Back Space	BS
17	Program Operator	POC[1]
24	Inhibit Presentation	INP
25	Line Feed	LF
2B	Format	FMT[1]
34	Presentation Position	PP[1]
35	Transparent	TRN[1]

Note:

1. Functions with parameters

Parameters in SCS can be one of two types:

- *Function parameters* extend the function defined by the function code. For example, the PP control function has a function parameter to explicitly define the positioning function performed. The form for a function parameter is a single EBCDIC character.
- *Value parameters* specify a numeric value for the function. For example, the PP function also has a value parameter for it. If the move is relative to the current position, the value parameter specifies the number of columns or lines the presentation position is to be moved from its current position. If the move is absolute, the value parameter specifies the absolute column or line number to which the presentation position is to move. The form for a value parameter is a 1-byte binary number.

Part 10. IMS compliance data access

The topic in this section describes IMS compliance control block and how to access data in the control blocks.

Chapter 52. IMS compliance control blocks

To make compliance audit data in IMS accessible for security audit compliance checks, IMS compliance control blocks are provided in IMS Operations Manager and IMS Connect address spaces.

Some IMS address spaces that use IMS Base Primitive Environment (BPE) provide compliance-related data in in-memory control blocks that are shipped as source within the address spaces. This compliance-related data is consolidated from other internal control blocks whose mappings are not shipped as source. To allow products that perform security audit compliance checks to obtain compliance audit data, IMS compliance control blocks are provided in the following IMS address spaces:

- IMS Operations Manager (OM)
- IMS Connect

IMS compliance control blocks are classified as a DMTI (Diagnosis, Modification, and Tuning Interface).

Structure of the BPE address space compliance data

BPE creates a primary-level address space z/OS name/token pair at the initialization of address spaces that provide compliance data. The name of the name/token pair is the same for all address spaces that provide compliance data. The name is a set value: BPECOMPLIANCEDAT.

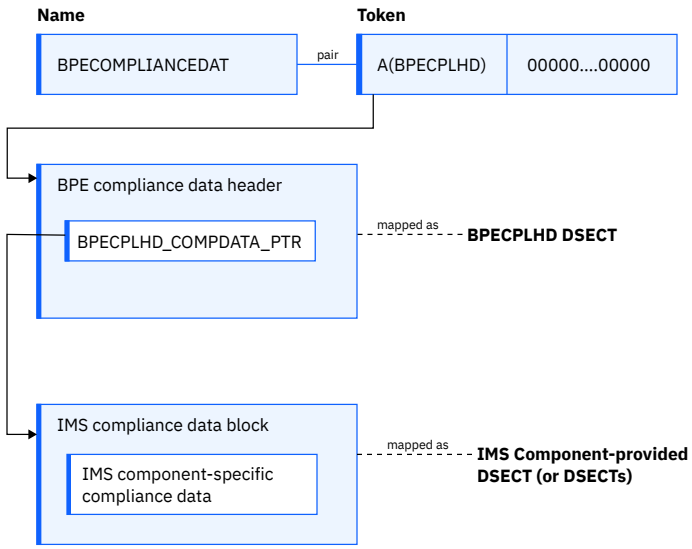


Figure 29. General structure of BPE address space compliance data

The first four bytes of the token in the pair are the 31-bit address of a BPE compliance data header, which is mapped by the source-shipped BPECPLHD macro as **DSECT BPECPLHD**. The BPECPLHD macro contains general system information about the address space, such as the address space type, name, and version.

The **BPECPLHD_COMPDATA_PTR** field points to a component-specific compliance data block that is unique to the component address space type, for example, OM or IMS Connect.

Some components might have extra compliance subblocks that are connected to the main compliance data block.

Locating IMS compliance data blocks

You can find the address of the BPECPLHD and the main component compliance data block by using the following code sample.

Restriction: The following code sample contains code fragments and is shown only for illustration purposes. This code can be part of a larger module for accessing the compliance data.

```
         L     R15,X'10'           Get A(CVT)
         L     R15,X'220'(,R15)    Follow chain per doc in
         L     R15,X'14'(,R15)      MVS Auth Assembler Ref
         L     R15,X'08'(,R15)     NT retrieve rtn address

         CALL  (15),(CPNTLEV,CPNTNAME,DS_TOKEN,DS_RETCODE),           *
               MF=(E,DS_PARMS)     Retrieve token

         ICM   R15,15,DS_RETCODE   Get IEANTRT return code
         JNZ   ERROR               If not 0, not found or error

         L     R9,DS_TOKEN         Token word 1=A(BPECPLHD)
         USING BPECPLHD,R9         Address BPECPLHD
         L     R8,BPECPLHD_COMPDATA_PTR  Pt to component data

       :
       :

*
* Module constants and literals
*
         DS    0F                  Ensure FW aligned
CPNTNAME DS    0CL16               NT name =
*                                   'BPECOMPLIANCEDAT'
         DC    A(BPECPLHD_NT_NAME1)  - 'BPEC'
         DC    A(BPECPLHD_NT_NAME2)  - 'OMPL'
         DC    A(BPECPLHD_NT_NAME3)  - 'IANC'
         DC    A(BPECPLHD_NT_NAME4)  - 'EDAT'

CPNTLEV  DC    A(IEANT_PRIMARY_LEVEL)  Name token level

         LTORG ,                   Literals

          :
          :

*
* DSECT mapping module working storage (assumed based via
* a USING in the code above)
*
DYNSTOR       DSECT ,
DS_TOKEN      DS    CL16          Retrieved name token
DS_RETCODE    DS    F             Return code from services

DS_PARMS      DS    0F
              DS    4F            Parmlist for IEANTRT

          :
          :

         BPECPLHD ,                Inc BPECPLHD DSECT
         IEANTASM ,                Inc z/OS name/token sym
```

Once you find the BPECPLHD block in an address space, perform the following tests to validate the blocks are ready for use:

1. Test the **BPECPLHD_CURSTCK** field or the **BPECPLHD_UPDCOUNT** field to check whether the value is zero.
   - If the value of the field is zero, the compliance data is not initialized yet. Don't use any data that is contained within the compliance blocks.
2. Test flag bits **BPECPLHD_F1_TERM** and **BPECPLHD_F1_ABTERM**.

- If any flag is set, the address space is in termination processing. Don't continue to access the **BPECPLHD** block or any related blocks because the storage that contains the blocks could be freed at any time.

3. Check the 4-character IMS component address space type in the **BPECPLHD_COMPTYPE** field to determine the kind of IMS address space data that you are accessing. This field is left-aligned and padded with blanks if needed.
   - The type is *OM* for an IMS Operations Manager address space.
   - The type is *HWS* for an IMS Connect address space.

Operations Manager compliance block

In an Operations Manager (OM) address space, the BPECPLHD_COMPDATA_PTR field points to the OM compliance data block. This block is defined in the CSLOCPLB macro as DSECT CLSOCPLB. The CSLOCPLB macro has one subblock that is mapped by DSECT CSLOCPLB_PLEX. The CSLOCPLB_PLEX contains compliance data that is specified in the IMSPLEX statement in the CSLOINxx PROCLIB member. The main CSLOCPLB block field CSLOCPLB_PLEXPTR points to the CSLOCPLB_PLEX block.

IMS Connect compliance block

In an IMS Connect address space, the BPECPLHD_COMPDATA_PTR field points to the IMS Connect compliance data block. This block is defined in the HWSCPLB macro as DSECT HWSCPLB.

The HWSCPLB has two chains of subblocks:

HWSCPLB_PLEX
Contains compliance data that is associated with an IMSplex, which is defined to the IMS Connect address space. The main HWSCPLB block field **HWSCPLB_PLEXPTR** points to the first **HWSCPLB_PLEX** block on the chain, and the value might be zero. The chain is a singly linked list that uses the **HWSCPLB_PLEX_NEXTPTR** field. A zero in this pointer indicates the end of chain.

HWSCPLB_DSTR
Contains compliance data that is associated with a data store, which is defined to the IMS Connect address space. The main HWSCPLB block field **HWSCPLB_DSTRPTR** points to the first **HWSCPLB_DSTR** block on the chain, and the value might be zero. The chain is a singly linked list that uses the **HWSCPLB_DSTR_NEXTPTR** field. A zero in this pointer indicates the end of chain.

IMS Connect allows both IMSplexes and data stores to be deleted via type 2 DELETE IMSCON commands. When an IMSplex or a data store is deleted, its corresponding HWSCPLB_PLEX or HWSCPLB_DSTR block is not removed from the HWSCPLB chain. Instead, IMS Connect marks the block as being logically deleted by setting a flag bit, HWSCPLB_PLEX_F1_DEL for IMSplex blocks, or HWSCPLB_DSTR_F1_DEL for data store blocks. If you write code that scans the IMSplex and data store compliance block chains, you should test the delete flag in each block, and skip processing it if the bit is set. Also note that IMS Connect will find and reuse a logically-deleted compliance block if a new IMSplex or data store is later created by a CREATE IMSCON command.

Field level details

For details on individual compliance data that is collected in each block, see the macros BPECPLHD, CSLOCPLB, and HWSCPLB in the IMS SDFSMAC macro data set.

Ensuring consistency and recovering compliance data

To ensure data consistency and recover data, save your data, set up recovery protection for asynchronous programs, and regularly retrieve the compliance token.

The IMS component code may update the compliance data at any time, including while your program is accessing the compliance blocks (for example, some settings can be changed dynamically by command). To ensure consistency of data, save the value in the **BPECPLHD_CURSTCK** field or the **BPECPLHD_UPDCOUNT** field before you access the compliance data. When you have the data that is

needed, verify whether the value in **BPECPLHD_CURSTCK** or **BPECPLHD_UPDCOUNT** is the same value that you saved. If the two values are different, gather the data again to ensure the consistency because the data is updated.

If your program is accessing either the BPECPLHD block or the related component compliance blocks while it is running asynchronously to the BPE address space, it must have a recovery set.

Remember: Running asynchronously in this case means running under:

- A service request block (SRB)
- A task control block (TCB) in the address space that is not the BPE JOBSTEP TCB or a descendant TCB
- A TCB in another address space in cross-memory or AR mode

Recovery protection is required because the BPE address space might terminate while your program is running.

Attention: This could delete the storage in the address space.

BPE deletes the compliance name/token pair at the start of both a normal and an abnormal termination. However, if you previously located the name/token pair, you might retrieve the data before BPE deletes it. For that reason, always retrieve the token every time your code runs and never reuse a saved BPECPLHD or compliance data block address for an extended period.

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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 grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation North Castle Drive, MD-NC119 Armonk, NY 10504-1785 US

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

Intellectual Property Licensing Legal and Intellectual Property Law IBM Japan, Ltd. 19-21, Nihonbashi-Hakozakicho, Chuo-ku Tokyo 103-8510, Japan

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 jurisdictions 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide 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 Director of Licensing IBM Corporation North Castle Drive, MD-NC119 Armonk, NY 10504-1785 US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

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

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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 actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate 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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

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_.

Programming interface information

The information in these topics is intended to help you customize IMS environments. This information documents General-use Programming Interface and Associated Guidance 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 topic or by a General-use programming interface label.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Linux® is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

To learn more, see IBM Privacy Statement.

Bibliography

This bibliography lists all of the publications in the IMS 15.3 library.

Title	Acronym
IMS Version 15.3 Application Programming	APG
IMS Version 15.3 Application Programming APIs	APR
IMS Version 15.3 Commands, Volume 1: IMS Commands A-M	CR1
IMS Version 15.3 Commands, Volume 2: IMS Commands N-V	CR2
IMS Version 15.3 Commands, Volume 3: IMS Component and z/OS Commands	CR3
IMS Version 15.3 Communications and Connections	CCG
IMS Version 15.3 Database Administration	DAG
IMS Version 15.3 Database Utilities	DUR
IMS Version 15.3 Diagnosis	DGR
IMS Version 15.3 Exit Routines	ERR
IMS Version 15.3 Installation	INS
IMS Version 15.3 Licensed Program Specifications	LPS
IMS Version 15.3 Messages and Codes, Volume 1: DFS Messages	MC1
IMS Version 15.3 Messages and Codes, Volume 2: Non-DFS Messages	MC2
IMS Version 15.3 Messages and Codes, Volume 3: IMS Abend Codes	MC3
IMS Version 15.3 Messages and Codes, Volume 4: IMS Component Codes	MC4
IMS Version 15.3 Operations and Automation	OAG
IMS Version 15.3 Release Planning	RPG
IMS Version 15.3 System Administration	SAG
IMS Version 15.3 System Definition	SDG
IMS Version 15.3 System Programming APIs	SPR
IMS Version 15.3 System Utilities	SUR

Index

Special Characters

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Product Number: 5635-A06
5655-DS5
5655-TM4