CICS Transaction Server for z/OS
Version 5 Release 6

XPI Function Reference

Note

Before using this information and the product it supports, read the information in Product Legal Notices.

This edition applies to the IBM® CICS® Transaction Server for z/OS®, Version 5 Release 6 (product number 5655-Y305655-BTA) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this PDF

This PDF is a reference of the XPI macro functions that can be used by global user exit programs to access some CICS services. The XPI functions are grouped according to their functional relationships, generally by CICS domain. To find out how to use these functions in programs, see the PDF called *Developing CICS System Programs*. Before CICS TS V5.4, the information in this PDF was in the *Customization Guide*.

For details of the terms and notation used in this book, see Conventions and terminology used in the CICS documentation in IBM Knowledge Center.

Date of this PDF

This PDF was created on May 28th 2020.

Chapter 1. Business application manager domain XPI function

The XPI provides one business application manager domain function. This is the DFHBABRX call INQUIRE_ACTIVATION.

The INQUIRE_ACTIVATION call

The INQUIRE_ACTIVATION function is provided on the DFHBABRX macro call. Use the INQUIRE_ACTIVATION call to obtain the activity name and the process type for the business transaction activity of the current transaction.

INQUIRE_ACTIVATION

```
DFHBABRX [CALL,]
    [CLEAR,]
    [IN,
    FUNCTION(INQUIRE_ACTIVATION),
    [TRANSACTION_TOKEN(name8),]]
    [RETURNED_ACTIVITYID(buffer_descriptor)]
    [RETURNED_PROCESS_NAME(buffer_descriptor)]
    [OUT,
    [ACTIVITY_NAME(name16)]
    [PROCESS_TYPE(name8)]
    RESPONSE (name1 | *),
    REASON (name1 | *)]
```

This command is threadsafe.

ACTIVITY_NAME(name16)
Returns the 16-character, user-assigned, name of the BTS activity.

PROCESS_TYPE(name8)
Returns the 8-character identifier of the type definition of the BTS process.

RETURNED_ACTIVITYID(buffer_descriptor)
Returns the 52-character, CICS-assigned, identifier of the BTS activity. RETURNED_ACTIVITYID is an output parameter (BAM returns it) and the data type is buffer, so the caller must supply an area to be used as a buffer as input to the call.

RETURNED_PROCESS_NAME(buffer_descriptor)
Returns the 36-character name of the BTS process. RETURNED_PROCESS_NAME is an output parameter (BAM returns it) and the data type is buffer, so the caller must supply an area to be used as a buffer as input to the call.

TRANSACTION_TOKEN(name8)
Specifies the transaction token for the task being inquired on.

RESPONSE and REASON values for INQUIRE_ACTIVATION

RESPONSE	*REASON*
OK	None
EXCEPTION	ACTIVITY_NOT_FOUND
DISASTER	None
INVALID	INVALID_BUFFER_LENGTH
KERNERROR	None

RESPONSE	***REASON***
PURGED	None

For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

Chapter 2. Directory domain XPI functions

The XPI provides directory domain functions that you can use to open and close an LDAP session, browse results for credentials, scan and locate results, close the browse, return the correct value and close the search.

The directory domain functions are the following DFHDDAPX calls:

- BIND_LDAP
- END_BROWSE_RESULTS
- FLUSH_LDAP_CACHE
- FREE_SEARCH_RESULTS
- GET_ATTRIBUTE_VALUE
- GET_NEXT_ATTRIBUTE
- GET_NEXT_ENTRY
- SEARCH_LDAP
- START_BROWSE_RESULTS
- UNBIND_LDAP

The BIND_LDAP call

The BIND_LDAP call establishes a session with an LDAP server.

The LDAP server is identified by one of the following:

- The LDAP URL and the distinguished name and password of the user authorized to extract the expected data.
- A RACF® profile in the LDAPBIND class that contains the LDAP URL and distinguished name and password. This is the preferred option, as you do not need to code LDAP credentials in your application.

BIND_LDAP

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(BIND_LDAP),
    {LDAP_BIND_PROFILE(block-descriptor)|
    LDAP_SERVER_URL((block-descriptor),DISTINGUISHED_NAME((block-descriptor),
    PASSWORD(block-descriptor),}
    [CACHE_SIZE(name4),CACHE_TIME_LIMIT(name4),]]
    [OUT,
    LDAP_SESSION_TOKEN(name4),
    [LDAP_RESPONSE(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

CACHE_SIZE(name4)
a fullword that specifies the number of bytes available for caching LDAP search results. A value of zero indicates an unlimited cache size. If CACHE_SIZE is specified, CACHE_TIME_LIMIT must also be specified. If neither parameter is specified, results will not be cached.

CACHE_TIME_LIMIT(name4)
a fullword that specifies the amount of time (in seconds) that LDAP search results are cached. A value of zero indicates an unlimited cache time limit.

DISTINGUISHED_NAME(block-descriptor)
specifies the location of the LDAP distinguished name, of the user permitted to bind to the chosen server. The block-descriptor is two fullwords of data, in which the first word contains the address of the data, and the second word contains the length in bytes of the data.

For more information on block-descriptors, see XPI syntax.

LDAP_BIND_PROFILE(block-descriptor)
specifies the location of the name of a RACF profile in the LDAPBIND class that contains the URL and credentials for the LDAP server being accessed. The block-descriptor is two fullwords of data, in which the first word contains the address of the data, and the second word contains the length in bytes of the data.

For more information on block-descriptors, see XPI syntax. You should specify either LDAP_BIND_PROFILE, or all three LDAP_SERVER_URL, DISTINGUISHED_NAME and PASSWORD parameters.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API, in response to receiving URL and user credentials.

LDAP_SERVER_URL(block-descriptor)
specifies the location of the LDAP URL (in the format ldap://server:port) of the LDAP server being accessed. If the colon and port number are omitted, the port defaults to 389. The block-descriptor is two fullwords of data, in which the first word contains the address of the data, and the second word contains the length in bytes of the data.

For more information on block-descriptors, see XPI syntax.

LDAP_SESSION_TOKEN(name4)
the name of the fullword token that specifies the LDAP connection.

PASSWORD(block-descriptor)
specifies the location of the password for the user identified in the DISTINGUISHED_NAME input. The block-descriptor is two fullwords of data, in which the first word contains the address of the data, and the second word contains the length in bytes of the data.

For more information on block-descriptors, see XPI syntax.

RESPONSE and REASON values for BIND_LDAP

RESPONSE	***REASON***
OK	None
EXCEPTION	INVALID_BUFFER_LENGTH
	INVALID_LDAP_PROFILE
	INVALID_LDAP_URL
	LDAP_INACTIVE
	NOTAUTH
	NOTFOUND
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

The END_BROWSE_RESULTS call

The END_BROWSE_RESULTS call allows you to end the browse session that was started by the START_BROWSE_RESULTS call.

END_BROWSE_RESULTS

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(END_BROWSE_RESULTS),
    SEARCH_TOKEN(name4),]
    [OUT,
    [LDAP_RESPONSE(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

SEARCH_TOKEN(name4)
the name of the fullword token that is returned by the SEARCH_LDAP function.

RESPONSE and REASON values for END_BROWSE_RESULTS

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_TOKEN
	INVALID_CALLING_SEQUENCE
	NOTFOUND
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

The FLUSH_LDAP_CACHE call

The FLUSH_LDAP_CACHE call removes the contents of all cached search responses for the specified LDAP connection.

FLUSH_LDAP_CACHE

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(FLUSH_LDAP_CACHE),
    LDAP_SESSION_TOKEN(name4),]
    [OUT,
    [LDAP_RESPONSE(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

LDAP_SESSION_TOKEN(name4)
the name of the fullword token that was returned by the BIND_LDAP function.

RESPONSE and REASON values for FLUSH_LDAP_CACHE

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_TOKEN
	LDAP_INACTIVE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

The FREE_SEARCH_RESULTS call

The FREE_SEARCH_RESULTS call releases all storage held by the SEARCH_LDAP function. The search results are terminated and the search token is invalidated. If the application does not call the FREE_SEARCH_RESULTS function, it is invoked by CICS when the task is terminated.

FREE_SEARCH_RESULTS

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(FREE_SEARCH_RESULTS),
    SEARCH_TOKEN(name4),]
    [OUT,
    [LDAP_RESPONSE(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

SEARCH_TOKEN(name4)
the name of the fullword token that is returned by the SEARCH_LDAP function.

RESPONSE and REASON values for FREE_SEARCH_RESULTS

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_TOKEN
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

The GET_ATTRIBUTE_VALUE call

You can use the GET_ATTRIBUTE_VALUE call to retrieve the value associated with an attribute returned by the SEARCH_LDAP call. An entry is an LDAP record, and an attribute is one element within an entry. The attribute can be returned by either the GET_NEXT_ATTRIBUTE function, or by specifying the name of the attribute.

GET_ATTRIBUTE_VALUE

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(GET_ATTRIBUTE_VALUE),
    SEARCH_TOKEN(name4),
    LDAP_ATTRIBUTE_NAME(block-descriptor),
    LDAP_ATTRIBUTE_VALUE(buffer-descriptor),
    [ATTRIBUTE_TYPE(name4),]
    [VALUE_ARRAY_POSITION(name4),]]
    [OUT,
    [LDAP_RESPONSE(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

ATTRIBUTE_TYPE(name4)
Specifies the keyword CHARACTER or BINARY, indicating the format of the attribute. If this parameter is not specified, a value of CHARACTER is assumed.

LDAP_ATTRIBUTE_NAME(block-descriptor)
Specifies the location of the LDAP attribute name. The block-descriptor is two fullwords of data, in which the first word contains the address of the attribute name, and the second word contains the length in bytes of the attribute name. For more information on block-descriptors, see XPI syntax.

LDAP_ATTRIBUTE_VALUE(buffer-descriptor)
Indicates the buffer where you want the attribute value returned. A group of three fullwords are specified for the buffer-descriptor:

- The address where the result is returned.
- The maximum size in bytes, of the data returned.
- The actual length in bytes of the result. This can be specified as *, and the length is then returned in DDAP_LDAP_ATTRIBUTE_VALUE_N.

For more information on buffer-descriptors, see XPI syntax.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

SEARCH_TOKEN(name4)
the name of the fullword token that is returned by the SEARCH_LDAP function.

VALUE_ARRAY_POSITION(name4)
Specifies the position of the requested value, in the value array for the current attribute. This parameter is only required if multiple values are expected. Array indexing starts at position 1.

RESPONSE and REASON values for GET_ATTRIBUTE_VALUE

RESPONSE	REASON
OK	None
EXCEPTION	INVALID_TOKEN
	NOTFOUND

RESPONSE	REASON
	INVALID_BUFFER_LENGTH
	INVALID_CALLING_SEQUENCE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

The GET_NEXT_ATTRIBUTE call

The GET_NEXT_ATTRIBUTE call allows you to get the next attribute in a series, from an entry returned by the SEARCH_LDAP call. An entry is an LDAP record, and an attribute is one element within an entry.

GET_NEXT_ATTRIBUTE

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(GET_NEXT_ATTRIBUTE),
    SEARCH_TOKEN(name4),
    LDAP_ATTRIBUTE_NAME(buffer-descriptor),]
    [OUT,
    [LDAP_RESPONSE(name4),]
    [VALUE_COUNT(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

LDAP_ATTRIBUTE_NAME(buffer-descriptor)
indicates the buffer where you want the attribute name returned. A group of three fullwords are specified for the buffer-descriptor:

- The address where the data is returned.
- The maximum size in bytes, of the data returned.
- The actual length in bytes of the data. This can be specified as *, and the length is then returned in DDAP_LDAP_ATTRIBUTE_NAME_N.

For more information on buffer-descriptors, see XPI syntax.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

SEARCH_TOKEN(name4)
the name of the fullword token that is returned by the SEARCH_LDAP function.

VALUE_COUNT(name4)
a fullword containing the number of values returned for this attribute. There is usually one value returned.

RESPONSE and REASON values for GET_NEXT_ATTRIBUTE

RESPONSE	*REASON*
OK	None
EXCEPTION	BROWSE_END

RESPONSE	*REASON*
	INVALID_BUFFER_LENGTH
	INVALID_CALLING_SEQUENCE
	INVALID_TOKEN
	NOT_FOUND
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

The GET_NEXT_ENTRY call

The GET_NEXT_ENTRY call allows you to get the next entry, from a series of entries returned by the SEARCH_LDAP call. An entry is an LDAP record. The distinguished name associated with the entry is returned by this call.

GET_NEXT_ENTRY

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(GET_NEXT_ENTRY),
    SEARCH_TOKEN(name4),
    [DISTINGUISHED_NAME(buffer-descriptor),]]
    [OUT,
    [LDAP_RESPONSE(name4),]
    [ATTRIBUTE_COUNT(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

ATTRIBUTE_COUNT(name4)
specifies the number of attributes in the retrieved entry.

DISTINGUISHED_NAME(buffer-descriptor)
indicates the buffer where you want the distinguished name of the next entry in the search returned. A group of three fullwords are specified for the buffer-descriptor:

- The address where the data is returned.
- The maximum size in bytes, of the data is returned.
- The actual length in bytes of the data. This can be specified as *, and the length is then returned in DDAP_DISTINGUISHED_NAME_N.

For more information on buffer-descriptors, see XPI syntax.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

SEARCH_TOKEN(name4)
the name of the fullword token that is returned by the SEARCH_LDAP function.

RESPONSE and REASON values for GET_NEXT_ENTRY

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_TOKEN
	INVALID_BUFFER_LENGTH
	INVALID_CALLING_SEQUENCE
	BROWSE_END
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

The SEARCH_LDAP call

The SEARCH_LDAP call sends a search request to a specified LDAP server. The search specifies an LDAP distinguished name, that is the target of the search.

The search returns a series of results (attributes or entries) that can be browsed or selected. An entry is an LDAP record, and an attribute is one element within an entry.

SEARCH_LDAP

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(SEARCH_LDAP),
    LDAP_SESSION_TOKEN(name4),
    DISTINGUISHED_NAME(block-descriptor),
    [FILTER(block-descriptor),]
    [SEARCH_TIME_LIMIT(name4),]]
    [OUT,
    SEARCH_TOKEN(name4),
    [LDAP_RESPONSE(name4),]
    [ENTRY_COUNT(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

DISTINGUISHED_NAME(block-descriptor)
specifies the location of the LDAP distinguished name. The block-descriptor is two fullwords of data, in which the first word contains the address of the data, and the second word contains the length in bytes of the data. For more information on block-descriptors, see XPI syntax.

ENTRY_COUNT(name4)
the number of LDAP entries returned by the search.

FILTER(block-descriptor)
specifies the location of an LDAP filter string that limits the search. If this parameter is not specified or is zero, the search filter is set to `(objectClass=*)`. The block-descriptor is two fullwords of data, in which the first word contains the address of the data, and the second word contains the length in bytes of the data. For more information on block-descriptors, see XPI syntax.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

LDAP_SESSION_TOKEN(name4)
the name of the fullword token that was returned by the BIND_LDAP function.

SEARCH_TIME_LIMIT(name4)
specifies the time limit for the search (in seconds). If the search is not successful within this time limit, the search is abandoned. If this parameter is not specified or is zero, the search time is unlimited.

SEARCH_TOKEN(name4)
the name of the fullword token that identifies and holds the current position in the search.

RESPONSE and REASON values for SEARCH_LDAP

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_BUFFER_LENGTH
	INVALID_TOKEN
	NOTFOUND
	TIMED_OUT
	LDAP_INACTIVE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

The START_BROWSE_RESULTS call

The START_BROWSE_RESULTS call allows you to browse the results (attributes or entries) returned by the SEARCH_LDAP call. START_BROWSE_RESULTS starts the browse at the first or only entry returned (there may be multiple entries returned by the search). The GET_NEXT_ENTRY call allows you to retrieve other entries.

START_BROWSE_RESULTS can be issued more than once for a SEARCH_TOKEN. If the call is issued after a GET_NEXT_ENTRY or GET_NEXT_ATTRIBUTE call, the browse position will be reset to the start of the search results.

START_BROWSE_RESULTS

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(START_BROWSE_RESULTS),
    SEARCH_TOKEN(name4),
    [DISTINGUISHED_NAME(buffer-descriptor),]]
    [OUT,
    [LDAP_RESPONSE(name4),]
    [ATTRIBUTE_COUNT(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

DISTINGUISHED_NAME(buffer-descriptor)
indicates the buffer where you want the distinguished name of the first, or only located result returned. A group of three fullwords are specified for the buffer-descriptor:

- The address where the data is returned.
- The length of the buffer in bytes, where the data is returned.
- The maximum length in bytes of the data. This can be specified as *, and the length is then returned in DDAP_DISTINGUISHED_NAME_N.

For more information on buffer-descriptors, see XPI syntax.

ATTRIBUTE_COUNT(name4)
a fullword indicating the number of attributes that can be browsed in the current entry.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

SEARCH_TOKEN(name4)
the name of the fullword token that is returned by the SEARCH_LDAP function.

RESPONSE and REASON values for START_BROWSE_RESULTS

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_TOKEN
	INVALID_BUFFER_LENGTH
	INVALID_CALLING_SEQUENCE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

The UNBIND_LDAP call

The UNBIND_LDAP call terminates a session with an LDAP server.

UNBIND_LDAP

```
DFHDDAPX [CALL],
    [CLEAR],
    [IN,
    FUNCTION(UNBIND_LDAP),
    LDAP_SESSION_TOKEN(name4),]
    [OUT,
    [LDAP_RESPONSE(name4),]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

This command is threadsafe.

LDAP_RESPONSE(name4)
specifies the return code that is sent by the LDAP API.

LDAP_SESSION_TOKEN(name4)
the name of the fullword token that was returned by the BIND_LDAP function.

RESPONSE and REASON values for UNBIND_LDAP

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_TOKEN
	LDAP_INACTIVE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

Chapter 3. Dispatcher XPI functions

The XPI provides six dispatcher functions. These functions are the DFHDSSRX calls ADD_SUSPEND, SUSPEND, RESUME, DELETE_SUSPEND, and WAIT_MVS, and the DFHDSATX call CHANGE_PRIORITY.

Use of these dispatcher calls is limited. Check the details supplied for each exit in Global user exit programs before using any functions.

Note:

1. You must issue an ADD_SUSPEND call to create a suspend token **before** you issue a SUSPEND or RESUME call.
2. If a suspended task is canceled, the SUSPEND fails with a RESPONSE value of 'PURGED' and a REASON value of 'TASK_CANCELLED'. A corresponding RESUME call returns with a RESPONSE value of 'EXCEPTION' and a REASON value of 'TASK_CANCELLED'.
3. If a suspended task is timed out, the SUSPEND fails with a RESPONSE value of 'PURGED' and a REASON value of 'TIMED_OUT'. A corresponding RESUME call returns with a RESPONSE value of 'EXCEPTION' and a REASON value of 'TIMED_OUT'.
4. Dispatcher protocols require that you issue a RESUME even if the SUSPEND was purged (due to task cancel or time out). You must issue only one RESUME for each SUSPEND call.

Synchronization protocols for SUSPEND and RESUME processing

If you use XPI SUSPEND and RESUME processing, you must observe the correct protocols, so that task purging can be handled effectively.

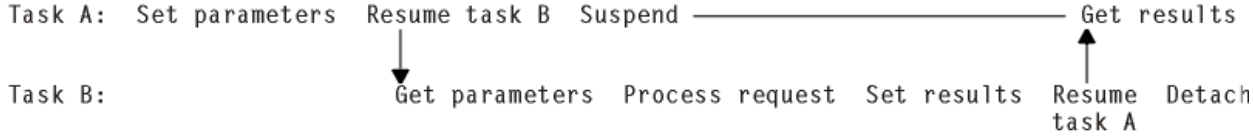
The normal synchronization protocol

In the normal case, synchronization involves two tasks and three operations.

In the following sample operations, the tasks are A (the task that requests a service) and B (the task that processes a request from task A).

1. Task A starts the request by:
   - Setting the parameters to be used by task B
   - Resuming task B
   - Issuing the SUSPEND call.
2. Task B performs the action by:
   - Getting the parameters
   - Performing the action
   - Setting the results
   - Terminating (or waiting for new work).
3. Task A ends the interaction by:
   - Getting the results left by task B.

This sequence looks like:



Ignoring the Resume and Suspend, the execution amounts to:

```
Set parameters; Get parameters; Process request; Set results; Get results
```

where these actions are always **sequential**.

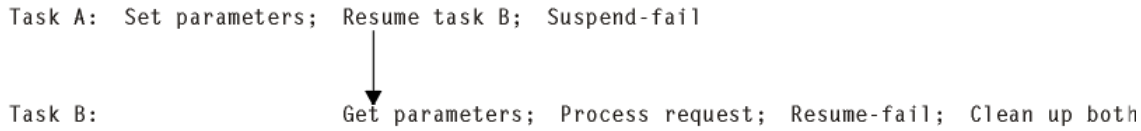
The synchronization protocol and task purge

If one of the tasks is to be purged, it is task A, because task A is the one suspended. In this case, execution of task A after the failed SUSPEND would be in parallel with task B; the proper serialization would be lost. If the program remained unchanged, Process request and Set results would be taking place at the same time as Get results, with unpredictable results.

Alternative approach to task purge

One way of preventing this problem is to ensure that task A, if it is to be purged, does not do anything that could interfere with task B. It might mean that A must not detach, if doing so releases storage that B needs to access. Because the only task that is now involved is task B, task B is left with the responsibility of cleaning up for both tasks.

The sequence is shown in the following diagram:



Because task-purging is effective only if performed between SUSPEND and RESUME, Suspend-fail precedes Resume-fail. With the same constraints on serialization as in the normal synchronization protocol, the task-purge protocol can be logically reduced to the following sequence:

```
Set parameters;  Get parameters;  Process request;  Clean up
```

The difference is that Set results and Get results are replaced by Clean up. It is vital that only these two sequences can happen; this means that both programs must be coded correctly. CICS ensures that both tasks are told either that SUSPEND and RESUME processing worked, or that it failed.

The following shows the programming steps that conform to these rules:

```
Program for Task A                  Program for Task B
SET PARAMETERS;
RESUME B;                           GET PARAMETERS;
SUSPEND A;                          PROCESS REQUEST;
                                    RESUME A;
if                                  if
  RESPONSE = OK                       RESPONSE ¬= OK
then                                then
  GET RESULTS;                        CLEAN UP;
endif                               endif
```

If both the SUSPEND and RESUME return ‘OK', the example follows the rules for the normal synchronization; processing finishes at Get results. If neither SUSPEND nor RESUME returns ‘OK', the example follows the rules for the task-purge protocol, and processing finishes at Clean up.

The sequence described previously is one method for dealing with the problem of task purge. Using this method, task B does not know, when it is processing the request, whether or not task A has been purged; this means that B must take great care in its use of resources owned by A (in case A has been purged). In some situations, this restriction may cause difficulties.

A different approach is as follows; if task A is to be purged:

1. A communicates to B that it is no longer available, thus informing B not to use any resources owned by A.
2. A performs its own clean-up processing (including issuing the RESUME call for the purged SUSPEND, as required by the dispatcher protocols), and abends.
3. B performs its own clean-up processing.

The ADD_SUSPEND call

ADD_SUSPEND acquires a suspend token that can later be used to identify a SUSPEND/RESUME pair.

ADD_SUSPEND

```
DFHDSSRX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(ADD_SUSPEND),
      [RESOURCE_NAME(name16 | string | 'string'),]
      [RESOURCE_TYPE(name8 | string | 'string'),]]
      [OUT,
      SUSPEND_TOKEN(name4 | (Rn)),
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

RESOURCE_NAME(name16 | string | "string")
specifies a 16-character string that can be used to document and trace the resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

name16
The name of the location where a 16-byte value is stored.

string
A string of characters without intervening blanks; if it is not 16 bytes long, it is extended with blanks or truncated as required.

"string"
A string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Note: RESOURCE_NAME on ADD_SUSPEND supplies a default value which is used if RESOURCE_NAME is not specified on a SUSPEND call.

RESOURCE_TYPE(name8 | string | "string")
specifies an 8-character string that can be used to document and trace the resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

name8
The name of the location where an 8-byte value is stored.

string
A string of characters without intervening blanks; if it is not 8 bytes long, it is extended with blanks or truncated as required.

"string"
A string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Note: RESOURCE_TYPE on ADD_SUSPEND supplies a default value which is used if RESOURCE_TYPE is not specified on a SUSPEND call.

SUSPEND_TOKEN(name4 | (Rn))
returns a token assigned by the system to identify the SUSPEND/RESUME pair of operations used on the task.

name4
The name of a 4-byte field where the token is stored

(Rn)
A register into which the token value is loaded.

RESPONSE and REASON values for ADD_SUSPEND

RESPONSE	*REASON*
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

The CHANGE_PRIORITY call

CHANGE_PRIORITY allows the issuing task to change its own priority. It cannot be used to change the priority of another task. This command causes the issuing task to release control, and so provide other tasks with the opportunity to run.

CHANGE_PRIORITY

```
DFHDSATX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(CHANGE_PRIORITY),
      PRIORITY(name1 | (Rn) | decimalint | literalconst),]
      [OUT,
      [OLD_PRIORITY(name1 | (Rn)),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

OLD_PRIORITY(name1 | (Rn))
returns the previous priority of the issuing task.

name1
The name of a 1-byte field where the task's previous priority is stored

(Rn)
A register in which the low-order byte receives the previous priority value and the other bytes are set to zero.

PRIORITY(name1 | (Rn) | decimalint | literalconst)
specifies the new priority to be assigned to the issuing task.

name1
The name of a 1-byte field, with a value in the range 0 through 255.

(Rn)
A register with the low-order byte containing the new priority value.

decimalint
A decimal integer not exceeding 255 in value. Neither an expression nor hexadecimal notation is allowed.

literalconst
A number in the form of a literal, for example B'00000000', X'FF', X'FCF4', "0" or an equate symbol with a similar value.

RESPONSE and REASON values for CHANGE_PRIORITY

RESPONSE	*REASON*
OK	None
DISASTER	None
INVALID	None
KERNERROR	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

The DELETE_SUSPEND call

DELETE_SUSPEND releases a suspend token associated with this task.

DELETE_SUSPEND

```
DFHDSSRX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(DELETE_SUSPEND),
      SUSPEND_TOKEN(name4 | (Rn)),]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

SUSPEND_TOKEN(name4 | (Rn))
specifies a token assigned by the system to identify the SUSPEND/RESUME pair of operations used on the task.

name4
The name of a 4-byte field, where the token obtained by an ADD_SUSPEND call has been stored

(Rn)
A register containing the token value previously obtained.

RESPONSE and REASON values for DELETE_SUSPEND

RESPONSE	*REASON*
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

The RESUME call

RESUME restarts execution of a task that is suspended or timed out.

There must be only one RESUME request for each SUSPEND. However, because this is an asynchronous interface, a SUSPEND can be received either before or after its corresponding RESUME. You must ensure that you keep account of the SUSPEND and RESUME requests issued from your exit program.

RESUME

```
DFHDSSRX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(RESUME),
      SUSPEND_TOKEN(name4 | (Rn)),
      [COMPLETION_CODE(name1 | (Rn)),]]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

COMPLETION_CODE(name1 | (Rn))
specifies a user-defined “reason for RESUME” code during suspend and resume processing.

name1
The name of a 1-byte area to receive the code

(Rn)
A register, in which the low-order byte contains the completion code and the other bytes are zero.

SUSPEND_TOKEN(name4 | (Rn))
specifies a token assigned by the system to identify the SUSPEND/RESUME pair of operations used on the task.

name4
The name of a location where you have a 4-byte token previously obtained as output from an ADD_SUSPEND call

(Rn)
A register containing the token value.

RESPONSE and REASON values for RESUME

RESPONSE	***REASON***
OK	None
EXCEPTION	TASK_CANCELLED
	TIMED_OUT
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note:

1. For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.
2. ‘TASK_CANCELLED’ means that the task was canceled by operator action while it was suspended, and that the suspend token is available for use.

The SUSPEND call

SUSPEND suspends execution of a running task.

Suspended tasks can be resumed in one of two ways. You can issue the XPI RESUME call, or the task is resumed automatically if the INTERVAL value that you specify on the DFHDSSRX macro expires. Suspended tasks can also be purged by the operator, or by an application, or by the deadlock timeout facility.

SUSPEND

```
DFHDSSRX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(SUSPEND),
      PURGEABLE(YES|NO),
      SUSPEND_TOKEN(name4 | (Rn)),
      [INTERVAL(name4 | (Rn)),]
      [RESOURCE_NAME(name16 | string | 'string'),]
      [RESOURCE_TYPE(name8 | string | 'string'),]
      [TIME_UNIT(SECOND|MILLI_SECOND),]
      [WLM_WAIT_TYPE,]]
      [OUT,
      [COMPLETION_CODE(name1 | (Rn)),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

COMPLETION_CODE (name1 | (Rn))
Returns a user-defined “reason for action” code during suspend and resume processing.

name1
The name of a 1-byte area to receive the code. The value in this field is user-defined, and is ignored by CICS.

(Rn)
A register in which the low-order byte contains the completion code and the other bytes are zero.

INTERVAL(name4 | (Rn))
Specifies in seconds or milliseconds the time after which the task is automatically resumed and given a RESPONSE value of PURGED and a REASON value of TIMED_OUT. The time unit used on the INTERVAL option depends on the setting of the TIME_UNIT option. The INTERVAL value overrides any timeout (DTIMOUT) value specified for the transaction.

name4
The name of a 4-byte area, which is interpreted as a binary fullword.

(Rn)
A register containing the interval value, a binary fullword.

PURGEABLE(YES|NO)
Specifies whether your code can cope with the request being abnormally terminated as a result of a purge. There are four types of purge, as shown in Table 1 on page 21. Specifying PURGEABLE(NO) tells the dispatcher:

- To reject any attempt to PURGE the task.
- To suppress the deadlock timeout (DTIMOUT) facility (if applicable to this task) for the duration of this request.

Table 1. SUSPEND call - RESPONSE(PURGED)

REASON	CONDITION	PURGEABLE (NO)	PURGEABLE (YES)
TASK_CANCELLED	PURGE	Canceled	Proceeds normally
	FORCEPURGE	Proceeds normally	Proceeds normally
TIMED_OUT	DTIMOUT	Canceled	Proceeds normally
	INTERVAL	Proceeds normally	Proceeds normally

Note: A FORCEPURGE always assumes that the user wants the task to be purged, and so overrides the PURGEABLE(NO) option. If the user has set an INTERVAL, then this, too, overrides the PURGEABLE(NO) option.

RESOURCE_NAME(name16 | string | "string")
Specifies a 16-character string that can be used to document and trace the resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

name16
The name of the location where a 16-byte value is stored.

string
A string of characters without intervening blanks; if it is not 16 bytes long, it is extended with blanks or truncated as required.

"string"
A string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Note:

1. CICS does not use the RESOURCE_NAME information but includes it in trace entries, and displays it on appropriate CEMT screens to help you to see what your task is doing. CICS internal requests specify values, and you should use different values to avoid ambiguity. CICS internal request values are described in The resources that CICS tasks can wait for in Troubleshooting.
2. If RESOURCE_NAME is not specified, the default value, if any, from ADD_SUSPEND is used.

RESOURCE_TYPE(name8 | string | "string")
Specifies an 8-character string that can be used to document and trace the resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

name8
The name of the location where an 8-byte value is stored.

string
A string of characters without intervening blanks; if it is not 8 bytes long, it is extended with blanks or truncated as required.

"string"
A string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Note:

1. CICS does not use the RESOURCE_TYPE information but includes it in trace entries, and displays it on appropriate CEMT screens to help you to see what your task is doing. CICS internal requests specify values, and you should use different values to avoid ambiguity. CICS internal request values are documented in The resources that CICS tasks can wait for in Troubleshooting.
2. If RESOURCE_TYPE is not specified, the default value, if any, from ADD_SUSPEND is used.

SUSPEND_TOKEN(name4 | (Rn))
Specifies a token assigned by the system to identify the SUSPEND/RESUME pair of operations used on the task.

name4
The name of a location where you have a 4-byte token previously obtained as output from an ADD_SUSPEND call

(Rn)
A register containing the token value.

TIME_UNIT(SECOND | MILLI_SECOND)
Specifies the time unit used on the INTERVAL option.

SECOND
The INTERVAL option specifies the number of seconds before timeout.

MILLI_SECOND
The INTERVAL option specifies the number of milliseconds before timeout.

WLM_WAIT_TYPE(name1)

Specifies, in a 1-byte location, the reason for suspending the task. This reason indicates the nature of the wait state to the MVS™ workload manager.

The equated values for the type of wait are as follows:

CMDRESP
Waiting on a command response.

CONV
Waiting on a conversation.

DISTRIB
Waiting on a distributed request.

IDLE
A CICS task, acting as a work manager, that has no work request that is allowed to service within the monitoring environment. For example, journaling code that suspends itself when there are no journaling I/O operations to perform.

IO
Waiting on an I/O operation or indeterminate I/O-related operation (locks, buffer, string, and so on).

LOCK
Waiting on a lock.

MISC
Waiting on an unidentified resource.

Note: This value is the default reason given to the wait if you suspend a task and do not specify the WLM_WAIT_TYPE parameter.

OTHER_PRODUCT
Waiting on another product to complete its function; for example, when the workload has been passed to Db2®.

SESS_LOCALMVS
Waiting on the establishment of a session in the MVS image on which this CICS region is running.

SESS_NETWORK
Waiting on the establishment of a session elsewhere in the network (that is, not on this MVS image).

SESS_SYSPLEX
Waiting on establishment of a session somewhere in the sysplex (that is, not on this MVS image).

TIMER
Waiting on the timeout of a timer (for example, a task that puts itself to sleep).

If you are running CICS in an MVS goal-mode workload management environment (that is, you are using goal-oriented performance management), specify the reason for suspending the task on the WLM_WAIT_TYPE parameter.

Table 2. RESPONSE and REASON values for SUSPEND

RESPONSE	REASON
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	TASK_CANCELLED

Table 2. RESPONSE and REASON values for SUSPEND (continued)	
RESPONSE	**REASON**
	TIMED_OUT

Note:

1. For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.
2. TASK_CANCELLED means that the task has been canceled by operator action or by an application command.
3. After a PURGED response, the suspend token must not be reused in another SUSPEND until it has been reset by a RESUME corresponding to the purged SUSPEND.
4. TIMED_OUT means that the task has been automatically resumed because the specified INTERVAL (or the timeout value specified at task attach) has expired. The token, however, remains suspended and must be the object of a RESUME before it can be the object of a DELETE_SUSPEND.

The WAIT_MVS call

WAIT_MVS requests a wait on an MVS event control block (ECB) or on a list of MVS ECBs. For example, you could issue the WAIT_MVS to wait for completion of an MVS task for which you have issued ATTACH and provided a task-completion ECB.

The dispatcher does not clear the ECBs when a WAIT_MVS request is received. If any ECB is already posted, control is returned immediately to the exit program with a response of ‘OK'.

A single ECB must not be the subject of more than one wait at a time. If any ECB is already being waited on when a WAIT_MVS request is received, the request is rejected. The RESPONSE code is ‘DSSR_INVALID', and the REASON code ‘DSSR_ALREADY_WAITING'.

Note: ECBs used in WAIT_MVS requests must always be posted using the MVS POST macro.

WAIT_MVS

```
DFHDSSRX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(WAIT_MVS),
      {ECB_ADDRESS(name4 | (Ra)) | ECB_LIST_ADDRESS(name4 | (Ra)),}
      PURGEABLE(YES|NO),
      [INTERVAL(name4 | (Rn)),]
      [RESOURCE_NAME(name16 | string | 'string'),]
      [RESOURCE_TYPE(name8 | string | 'string'),]]
      [TIME_UNIT(SECOND|MILLI_SECOND),]
      [WLM_WAIT_TYPE,]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

ECB_ADDRESS(name4 | (Ra))
Specifies the address of the ECB to be waited on.

name4
The name of a location that contains an ECB address.

(Ra)
A register that contains the address of an ECB.

ECB_LIST_ADDRESS(name4 | (Ra))
Specifies the address of a list of ECB addresses to be waited on.

name4
The name of a location that contains an ECB address, possibly followed by more ECB addresses. The last address word in the list has the high-order bit set to 1.

(Ra)
A register pointing to an address list as previously described.

INTERVAL(name4 | (Rn))
Specifies in seconds or milliseconds the time after which the task is automatically resumed and given a RESPONSE value of 'PURGED' and a REASON value of 'TIMED_OUT'. The time unit used on the INTERVAL option depends on the setting of the TIME_UNIT option.

The INTERVAL value overrides any timeout (DTIMOUT) value specified for the transaction.

name4
The name of a 4-byte area, which is interpreted as a binary fullword

(Rn)
A register containing the interval value, a binary fullword.

PURGEABLE(YES|NO)
Specifies whether your code can cope with the request being abnormally terminated as a result of a purge. There are four types of purge, as shown in Table 3 on page 25. Specifying PURGEABLE(NO) tells the dispatcher:

- To reject any attempt to PURGE the task
- To suppress the deadlock timeout (DTIMOUT) facility (if applicable to this task) for the duration of this request.

Table 3. SUSPEND call - RESPONSE(PURGED)

REASON	CONDITION	PURGEABLE (NO)	PURGEABLE (YES)
TASK_CANCELLED	PURGE	Canceled	Proceeds normally
	FORCEPURGE	Proceeds normally	Proceeds normally
TIMED_OUT	DTIMOUT	Canceled	Proceeds normally
	INTERVAL	Proceeds normally	Proceeds normally

Note: A FORCEPURGE always assumes that the user wants the task to be purged, and so overrides the PURGEABLE(NO) option. If the user has set an INTERVAL, then this, too, overrides the PURGEABLE(NO) option.

RESOURCE_NAME(name16 | string | "string")
Specifies a 16-character string that can be used to document and trace the resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

name16
The name of the location where a 16-byte value is stored.

string
A string of characters without intervening blanks; if it is not 16 bytes long, it is extended with blanks or truncated as required.

"string"
A string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Note: CICS does not use the RESOURCE_NAME information but includes it in trace entries, and displays it on appropriate CEMT screens to help you to see what your task is doing. CICS internal requests specify values, and you should use different values to avoid ambiguity. CICS internal request values are documented in The resources that CICS tasks can wait for in Troubleshooting.

RESOURCE_TYPE(name8 | string | "string")
Specifies an 8-character string that can be used to document and trace the resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

name
The name of the location where an 8-byte value is stored.

string
A string of characters without intervening blanks; if it is not 8 bytes long, it will be extended with blanks or truncated as required.

"string"
A string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Note: CICS does not use the RESOURCE_TYPE information but includes it in trace entries, and displays it on appropriate CEMT screens to help you to see what your task is doing. CICS internal requests specify values, and you should use different values to avoid ambiguity. CICS internal request values are documented in The resources that CICS tasks can wait for in Troubleshooting.

TIME_UNIT(SECOND | MILLI_SECOND)
Specifies the time unit used on the INTERVAL option.

SECOND
The INTERVAL option specifies the number of seconds before timeout.

MILLI_SECOND
The INTERVAL option specifies the number of milliseconds before timeout.

WLM_WAIT_TYPE(name1)
Specifies, in a 1-byte location, the reason for suspending the task. This indicates the nature of the task's wait state to the MVS workload manager.

The equated values for the type of wait are as follows:

CMDRESP
Waiting on a command response.

CONV
Waiting on a conversation.

DISTRIB
Waiting on a distributed request.

IDLE
A CICS task, acting as a work manager, that has no work request that is allowed to service within the monitoring environment. For example, journaling code that suspends itself when there are no journaling I/O operations to perform.

IO
Waiting on an I/O operation or indeterminate I/O-related operation (locks, buffer, string, and so on).

LOCK
Waiting on a lock.

MISC
Waiting on an unidentified resource. This is the default reason given to the wait if you suspend a task and do not specify the WLM_WAIT_TYPE parameter.

OTHER_PRODUCT
Waiting on another product to complete its function; for example, when the workload has been passed to Db2.

SESS_LOCALMVS
Waiting on the establishment of a session in the MVS image on which this CICS region is running.

SESS_NETWORK
Waiting on the establishment of a session elsewhere in the network (that is, not on this MVS image).

SESS_SYSPLEX
Waiting on establishment of a session somewhere in the sysplex (that is, not on this MVS image).

TIMER
Waiting on the timeout of a timer (for example, a task that puts itself to sleep).

If you are running CICS in an MVS goal-mode workload management environment (that is, you are using goal-oriented performance management), you are recommended to specify the reason for suspending the task on the WLM_WAIT_TYPE parameter.

Table 4. RESPONSE and REASON values for WAIT_MVS

RESPONSE	**REASON**
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	TASK_CANCELLED
	TIMED_OUT

Note:

1. For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.
2. TIMED_OUT is returned if the INTERVAL expires, or if a deadlock timeout interval expires.
3. TASK_CANCELLED means that the task has been canceled by operator action or by an application command.

Chapter 4. Dump control XPI functions

The XPI provides two dump control functions. These are the DFHDUDUX macro calls SYSTEM_DUMP and TRANSACTION_DUMP.

Restriction: DFHDUDUX calls cannot be used in any exit program invoked from any global user exit point in the:

- Statistics domain
- Monitor domain
- Dump domain
- Dispatcher domain
- Transient data program.

The SYSTEM_DUMP call

SYSTEM_DUMP causes a system dump to be taken. If the system dump code that you supply on input is in the system dump code table, the dump may be suppressed.

For information about the dump table and how it works, see Using dumps in problem determination and SET SYSDUMPCODE.

SYSTEM_DUMP

```
DFHDUDUX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(SYSTEM_DUMP),
      SYSTEM_DUMPCODE(name8 | string | "string"),
      [CALLER(block-descriptor),]
      [TITLE(block-descriptor),]]
      [OUT,
      DUMPID(name9 | *),
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

CALLER(block-descriptor)
specifies the source of a system dump request. The information that you supply here appears in the dump header, so you could use it to identify the exit program that initiated the system dump request. For a description of valid block-descriptors, see XPI syntax.

DUMPID(name9 | *)
returns the dump identifier.

name9
The name of a 9-byte field to receive the assigned ID.

SYSTEM_DUMPCODE(name8 | string | "string")
specifies the code corresponding to the error that caused this system dump call. System dump codes are held in the dump table.

name8
The name of a location containing an 8-byte string.

string
A string of characters without intervening blanks. The macro generates, from the string, a literal constant of length 8 bytes, extending with blanks or truncating as required.

"string"
A string, enclosed in quotation marks and possibly containing blanks. This value is processed in the same way as the preceding “string”.

TITLE(block-descriptor)
specifies an area containing the text you want to appear in the dump header when the system dump is printed.

RESPONSE and REASON values for SYSTEM_DUMP

RESPONSE	*REASON*
OK	None
EXCEPTION	FESTAE_FAILED
	INSUFFICIENT_STORAGE
	IWMWQWRK_FAILED
	NO_DATASET
	PARTIAL_SYSTEM_DUMP
	SDUMP_BUSY
	SDUMP_FAILED
	SDUMP_NOT_AUTHORIZED
	SUPPRESSED_BY_DUMPOPTION
	SUPPRESSED_BY_DUMPTABLE
	SUPPRESSED_BY_USEREXIT
DISASTER	None
INVALID	INVALID_DUMPCODE
	INVALID_PROBDESC
	INVALID_SVC_CALL
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

The TRANSACTION_DUMP call

TRANSACTION_DUMP causes a transaction dump to be taken. If the transaction dump code that you supply on input is in the transaction dump code table, the dump may be suppressed and, optionally, a system dump may be taken.

For information about the dump table and how it works, see Using dumps in problem determination and SET TRANDUMPCODE.

Valid characters include uppercase characters (A-Z), lowercase characters (a-z), digits (0-9), and the special characters $ @ # / % & ? ! : | ; , ¢ + * ¬ - and _. In some cases, the characters < > . = and " are also valid depending on where you set them. Any lowercase characters you enter are converted to uppercase.

Important

There is a restriction in using the XPI early during initialization. Do not start exit programs that use the XPI functions TRANSACTION_DUMP, WRITE_JOURNAL_DATA, MONITOR, and INQUIRE_MONITOR_DATA until the second phase of the PLTPI. For further information about the PLTPI, refer to Writing initialization and shutdown programs .

TRANSACTION_DUMP

```
DFHDUDUX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(TRANSACTION_DUMP),
      TRANSACTION_DUMPCODE(name4 | string | 'string')
      [CSA(NO|YES),]
      [PROGRAM(NO|YES),]
      [SEGMENT(block-descriptor),]
      [SEGMENT_LIST(block-descriptor),]
      [TCA(NO|YES),]
      [TERMINAL(NO|YES),]
      [TRANSACTION(NO|YES),]
      [TRT(NO|YES),]]
      [OUT,
      DUMPID(name9 | *),
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

Note: This command is **NOT** threadsafe.

CSA(NO|YES)
specifies whether the common system area (CSA) is to be included in the transaction dump. The default is NO.

DUMPID(name9 | *)
returns the dump identifier.

name9
The name of a 9-byte field to receive the assigned ID.

PROGRAM(NO|YES)
specifies whether all program storage areas associated with this task are to be included in the transaction dump. The default is NO.

SEGMENT(block-descriptor)
specifies the address and the length of a single block of storage that is to be dumped. See XPI syntax for a description of valid block-descriptors. SEGMENT and SEGMENT_LIST are mutually exclusive.

SEGMENT_LIST(block-descriptor)
specifies the address and length of a *set* of contiguous word pairs. The first word in each pair specifies the **length** in bytes of a storage segment to be dumped; the second word contains the **address** of the storage segment. The end of the list must be marked by a word containing X'FFFFFFFF'. SEGMENT and SEGMENT_LIST are mutually exclusive.

TCA(NO|YES)
specifies whether the task control area (TCA) is to be included in the transaction dump. The default is NO.

TERMINAL(NO|YES)
specifies whether all terminal storage areas associated with the task are to be included in the transaction dump. The default is NO.

TRANSACTION(NO|YES)
specifies whether all transaction storage areas associated with the task are to be included in the transaction dump. The default is NO.

TRANSACTION_DUMPCODE(name4 | string | "string")
specifies the code corresponding to the error that caused this transaction dump call. Transaction dump codes are held in the dump table.

name4
The name of a location containing a 4-byte string.

string
A string of characters without intervening blanks. The macro generates a literal constant of length 4 bytes from the string, extending with blanks or truncating as required.

"string"
A string, enclosed in quotation marks and possibly containing blanks. This value is processed in the same way as the preceding “string”.

TRT(NO|YES)
specifies whether the trace table (TRT) is to be included in the transaction dump. The default is NO.

RESPONSE and REASON values for TRANSACTION_DUMP

RESPONSE	*REASON*
OK	None
EXCEPTION	FESTAE_FAILED
	INSUFFICIENT_STORAGE
	IWMWQWRK_FAILED
	NOT_OPEN
	OPEN_ERROR
	PARTIAL_SYSTEM_DUMP
	PARTIAL_TRANSACTION_DUMP
	SDUMP_BUSY
	SDUMP_FAILED
	SDUMP_NOT_AUTHORIZED
	SUPPRESSED_BY_DUMPOPTION
	SUPPRESSED_BY_DUMPTABLE
	SUPPRESSED_BY_USEREXIT
DISASTER	None
INVALID	INVALID_DUMPCODE
	INVALID_PROBDESC
	INVALID_SVC_CALL
KERNERROR	None
PURGED	None

Note:

1. For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.
2. NOT_OPEN means that the CICS dump data set is closed.
3. OPEN_ERROR means that an error occurred while a CICS dump data set was being opened.
4. PARTIAL means that the transaction dump resulting from this request is incomplete.

Chapter 5. Enqueue domain XPI functions

The XPI provides two enqueue domain functions. These are the DFHNQEDX calls DEQUEUE and ENQUEUE.

The DEQUEUE function

The DEQUEUE function is provided on the DFHNQEDX macro call. It releases a resource previously enqueued by an ENQUEUE function call.

DEQUEUE

```
DFHNQEDX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(DEQUEUE),
      {ENQUEUE_TOKEN(name4),|
      ENQUEUE_NAME1(address,length),[ENQUEUE_NAME2(address,length),]}
      MAX_LIFETIME(DISPATCHER_TASK),]
     [ENQUEUE_TYPE (XPI | EXECSTRN | EXECADDR),]
      [OUT,
      RESPONSE (name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

The ENQUEUE_TOKEN, ENQUEUE_NAME1, ENQUEUE_NAME2, MAX_LIFETIME (DISPATCHER_TASK), and ENQUEUE_TYPE (XPI | EXECSTRN | EXECADDR) parameters are the same as in the ENQUEUE function call.

RESPONSE and REASON values for DEQUEUE

RESPONSE	*REASON*
OK	None
EXCEPTION	ENQUEUE_NOT_OWNED ENQUEUE_LOCKED

The ENQUEUE function

The ENQUEUE function is provided on the DFHNQEDX macro call. It allows you to enqueue on a named resource.

By default, all enqueues created by XPI ENQUEUE commands are allocated to a specific enqueue pool called DISPATCH and are treated as internal to CICS. XPI enqueues do not conflict with enqueues created by **EXEC CICS ENQ** commands, which are added to different enqueue pools, depending on the enqueue model specified. For example, an active EXEC CICS ENQ on a string does not prevent an XPI ENQUEUE command on the same string being obtained.

Note:

- XPI enqueues cannot be browsed using the CICS SPI.
- XPI enqueues cannot be controlled by the use of ENQMODELs.

When you use the optional **ENQUEUE_TYPE** parameter, the XPI ENQUEUE command can enqueue on the same resource being enqueued on by EXEC CICS ENQ or vice versa. Applications can synchronize processes using EXEC CICS and EXEC XPI commands.

ENQUEUE

```
DFHNQEDX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(ENQUEUE),
      ENQUEUE_NAME1(address,length),
      [ENQUEUE_NAME2(address,length),]
      MAX_LIFETIME(DISPATCHER_TASK),
     [ENQUEUE_TYPE (XPI | EXECSTRN | EXECADDR),]
      [WAIT(YES|NO),]
      [PURGEABLE(YES|NO),]]
      [OUT,
      [ENQUEUE_TOKEN(name4),]
      [DUPLICATE_REQUEST,]
      RESPONSE (name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

DUPLICATE_REQUEST
Indicates that the requesting dispatcher task already owns the resource being enqueued.

ENQUEUE_NAME1(address,length)
Specifies the high-order part of the name to be enqueued.

ENQUEUE_NAME2(address,length)
Specifies the low-order part, if any, of the name to be enqueued.

ENQUEUE_TOKEN(name4)
Enables a subsequent DEQUEUE request to identify the resource by a token rather than enqueue name, allowing the NQ domain to locate the enqueue control block directly, and hence more efficiently.

ENQUEUE_TYPE (XPI | EXECSTRN | EXECADDR)
Specifies the type of resource being enqueued on. The XPI option specifies the typical DFHNQEDX behavior. The resource pool used is exclusive to XPI and cannot be accessed by the CICS API. Use EXECSTRN or EXECADDR to indicate that ENQUEUE_NAME1 specifies an enqueue resource, located in the same namespace as the one being used by EXEC CICS ENQ. For more information about EXECSTRN and EXECADDR, see The resources that CICS tasks can wait for in Troubleshooting.

MAX_LIFETIME(DISPATCHER_TASK)
MAX_LIFETIME(DISPATCHER_TASK) is required and specifies that all XPI enqueues are owned by the requesting dispatcher task.

If you use the ENQUEUE XPI call to ensure that your global user exit programs are threadsafe, you are recommended to free (dequeue) resources during the invocation of the global user exit program in which they were enqueued. However, because no recovery services are provided for stopping global user exits, CICS ensures that any outstanding XPI enqueues are dequeued automatically when the dispatcher task ends. If the dispatcher task is running a CICS transaction, the dispatcher task ends when the CICS transaction ends, whether normally or abnormally.

Usually, enqueues are owned by the requesting transaction, which contains units of work (UOWs), and these are used to anchor the enqueue control blocks. The XPI, however, does not require a transaction environment, and global user exits can be called under dispatcher tasks that have no transactions or UOWs.

PURGEABLE(YES|NO)
Specifies whether a purge (or timeout) request against the task is to be honored if the requesting dispatcher task has to wait for the enqueue.

WAIT(YES|NO)
Specifies whether the dispatcher task is to wait if the resource is currently enqueued to another dispatcher task.

RESPONSE and REASON values for ENQUEUE

RESPONSE	REASON
OK	None
EXCEPTION	ENQUEUE_BUSY ENQUEUE_LOCKED ENQUEUE_DISABLED LIMIT_EXCEEDED SYSENQ_FAILURE INVALID_PHASE
PURGED	TASK_CANCELLED TIMED_OUT

Chapter 6. Kernel domain XPI functions

The XPI provides two kernel domain functions. These are the DFHKEDSX calls START_PURGE_PROTECTION and STOP_PURGE_PROTECTION.

The START_PURGE_PROTECTION function

The START_PURGE_PROTECTION function is provided on the DFHKEDSX macro call. It inhibits purge, but not force-purge, for the current task. This function can be used by all global user exit programs to inhibit purge during a global user exit call.

In general, each START_PURGE_PROTECTION call should have a corresponding STOP_PURGE_PROTECTION function call to end the purge protection period on completion of any program logic that needs such protection.

START_PURGE_PROTECTION

```
DFHKEDSX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(START_PURGE_PROTECTION),]
       [OUT,
       RESPONSE (name1 | *)]
```

This command is threadsafe.

There are no input or output parameters on this call, only a RESPONSE.

The STOP_PURGE_PROTECTION function

The STOP_PURGE_PROTECTION function is provided on the DFHKEDSX macro call. It is re-enables purge for the current task after purge has been suspended by a preceding START_PURGE_PROTECTION function call.

STOP_PURGE_PROTECTION

```
DFHKEDSX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(STOP_PURGE_PROTECTION),]
       [OUT,
       RESPONSE (name1 | *)]
```

This command is threadsafe.

There are no input or output parameters on this call, only a RESPONSE.

Nesting purge protection calls

The START_ and STOP_PURGE_PROTECTION functions can be nested. You should ensure that, if multiple START_PURGE_PROTECTION calls are issued for a task, that the correct number of STOP_PURGE_PROTECTION calls are issued to cancel the purge protection.

If you issue two starts and only one stop, purge protection remains on for the current task.

For example, for any current task, more than one global user exit program may be driven. You must design your exit programs to ensure that purge protection is correctly cancelled. An example of nesting is shown as follows:

```
 XEIIN:
  EXIT_PROG1: Calls START_PURGE_PROTECTION

    XFCREQ:
```

```
     EXIT_PROG2: Calls START_PURGE_PROTECTION

    XFCREQC:
     EXIT_PROG3: Calls STOP_PURGE_PROTECTION

XEIOUT:
 EXIT_PROG4: Calls STOP_PURGE_PROTECTION
```

Chapter 7. Loader XPI functions

The XPI provides five loader functions. These functions are the DFHLDLDX calls ACQUIRE_PROGRAM, DEFINE_PROGRAM, DELETE_PROGRAM, IDENTIFY_PROGRAM, and RELEASE_PROGRAM.

The CICS loader services, including the XPI, recognize non-Language Environment® (LE) assembler programs that are linked AMODE(64). The addressing mode of the module is shown in the returned entry point parameter. AMODE(64) is indicated when bit 0 is 0 and bit 31 is 1 (the same addressing mode convention that is used in the z/OS operating system).

Restriction: DFHLDLDX calls cannot be used in any exit program invoked from any global user exit point in the following domains or program:

- Statistics domain
- Monitor domain
- Dump domain
- Dispatcher domain
- Transient data program

The ACQUIRE_PROGRAM call

ACQUIRE_PROGRAM returns the entry and load point addresses, the length, and a new program token for a usable copy of the named program, which can be identified by either its name or a program token.

ACQUIRE_PROGRAM

```
DFHLDLDX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(ACQUIRE_PROGRAM),
      {PROGRAM_NAME(name8 | string | 'string')|
      PROGRAM_TOKEN(name8)},
      [SUSPEND(NO|YES),]]
      [OUT,
      ENTRY_POINT(name4 | (Ra)),
      [LOAD_POINT(name4 | (Ra)),]
      [NEW_PROGRAM_TOKEN(name8),]
      [PROGRAM_ATTRIBUTE(name1 | (Rn)),]
      [PROGRAM_LENGTH(name4 | (Rn)),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

ENTRY_POINT(name4 | (Ra))
Returns the program's entry point address.

name4
The name of a 4-byte location to receive the 31-bit entry address

(Ra)
A register to receive the entry address.

LOAD_POINT(name4 | (Ra))
Returns the program's load point address.

name4
The name of a 4-byte location to receive the loaded address

(Ra)
A register that is to contain the load address.

NEW_PROGRAM_TOKEN(name8)
Returns the new program token for a usable copy of the named program.

name8
The name of a location to receive the 8-byte token that identifies this program and instance.

PROGRAM_ATTRIBUTE(name1 | (Rn))
Returns the program attribute.

name1
The name of a 1-byte location to receive the program attribute.

(Rn)
A register in which the low-order byte receives the program attribute and the other bytes are set to zero. It can have the values RELOAD, RESIDENT, REUSABLE, or TRANSIENT.

RELOAD
The program is not reusable, and therefore several copies of the program may be loaded. A copy is removed from storage when a RELEASE_PROGRAM call (for that copy) is issued.

RESIDENT
There is a single copy of the program that is not removed from storage unless deleted. RESIDENT programs must be at least quasireentrant. Any program of PROGRAM_TYPE SHARED has the RESIDENT attribute by default. The DELETE_PROGRAM call has no effect on this type of RESIDENT program.

REUSABLE
Similar to RESIDENT, except that a REUSABLE program that is not in use can be removed from storage by CICS, for storage optimization reasons.

TRANSIENT
Similar to RESIDENT, except that a TRANSIENT program is removed from storage as soon as its use count drops to zero.

PROGRAM_LENGTH(name4 | (Rn))
Returns the length of the named program.

name4
The name of a 4-byte location that is to receive the length in bytes, expressed in binary

(Rn)
A register to contain the length in bytes, expressed in binary.

PROGRAM_NAME(name8 | string | "string")
Specifies the name of the program to be acquired.

name8
The name of a location containing an 8-byte program name.

string
A string of characters naming the program.

"string"
A string in quotation marks. The string length is set to 8 by padding with blanks or truncating.

PROGRAM_TOKEN(name8),
Specifies a token identifying the program whose details are to be acquired.

name8
The name of a location containing the 8-byte token obtained from a previous DEFINE_PROGRAM or ACQUIRE_PROGRAM call.

SUSPEND(NO|YES)
Specifies whether execution is to be suspended until the request can be granted.

RESPONSE and REASON values for ACQUIRE_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	NO_STORAGE

RESPONSE	*REASON*
	PROGRAM_NOT_DEFINED
	PROGRAM_NOT_FOUND
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note:

1. For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.
2. A REASON of 'NO_STORAGE' with a RESPONSE of 'EXCEPTION' means that there was insufficient storage to satisfy this request, and SUSPEND(NO) was specified.
3. A REASON of 'PROGRAM_NOT_FOUND' is returned if the program has not been included in the library concatenation, or if the link-edit failed. In such a case, the program is marked as "not executable"; it must be re-linked before it can be successfully acquired.

The DEFINE_PROGRAM call

You can use the DEFINE_PROGRAM call to define new programs to the loader domain, or to change the details of programs that are already defined. The details that you provide are recorded on the local catalog, and become available immediately. They are used on all subsequent ACQUIRE requests for the named program.

Program definitions made using DEFINE_PROGRAM are not retained over an XRF takeover. Also, the CSD is not updated, only the loader domain definitions.

DEFINE_PROGRAM

```
DFHLDLDX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(DEFINE_PROGRAM),
      PROGRAM_NAME(name8 | string | 'string' ),
      [EXECUTION_KEY(CICS|USER),]
      [PROGRAM_ATTRIBUTE(RELOAD|RESIDENT|REUSABLE|TRANSIENT),]
      [PROGRAM_TYPE(PRIVATE|SHARED|TYPE_ANY),]
      [REQUIRED_AMODE(24|31|AMODE_ANY|64),]
      [REQUIRED_RMODE(24|RMODE_ANY),]]
      [OUT,
      [NEW_PROGRAM_TOKEN(name8),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

EXECUTION_KEY(CICS|USER)
Specifies, in conjunction with other program attributes, the type of dynamic storage area (DSA) into which the loader is to load the program.

CICS
For non-reentrant programs, the program is to be loaded into a CICS DSA above or below the 16 MB line; that is, the CDSA or ECDSA. The choice of CICS DSA depends on the residence mode (RMODE) attribute of the program, as defined to the linkage-editor.

For reentrant RMODE(24) programs, the program is to be loaded into the CDSA.

USER
For non-reentrant programs, the program is to be loaded into a user DSA above or below the 16 MB line; that is, the UDSA or EUDSA. The choice of user DSA depends on the residence mode (RMODE) attribute of the program, as defined to the linkage-editor.

For reentrant RMODE(24) programs, the program is to be loaded into the UDSA.

Reentrant programs eligible to reside above the 16 MB line: If a program is link-edited as reentrant with AMODE(31),RMODE(ANY), the EXECUTION_KEY option is ignored, and it is loaded into a read-only DSA (the RDSA or ERDSA). For details of the type of storage allocated for the ERDSA, see the RENTPGM system initialization parameter.

See Table 5 on page 42 for a summary of the effect of the EXECUTION_KEY option in conjunction with other factors.

Table 5. Summary of attributes defining DSA eligibility

EXECUTION_KEY option	Reentrant	Above or below 16 MB line	Dynamic storage area (DSA)
CICS	No	Below	CDSA
CICS	Yes	Below	RDSA
CICS	No	Above	ECDSA
CICS	Yes	Above	ERDSA
USER	No	Below	UDSA
USER	Yes	Below	RDSA
USER	No	Above	EUDSA
USER	Yes	Above	ERDSA

NEW_PROGRAM_TOKEN(name8)
Returns the token supplied for the newly-defined program.

name8
The name of a location to contain the 8-byte token obtained.

PROGRAM_ATTRIBUTE(RELOAD|RESIDENT|REUSABLE|TRANSIENT)
Specifies the residency status of the program.

RELOAD
Every ACQUIRE_PROGRAM request for this program is satisfied by loading a new copy into storage. When a RELEASE request is issued for a copy, it is removed from storage.

Note: Do not use this attribute when defining an exit program.

RESIDENT
There is a single copy of the program that is not removed from storage unless it is deleted. RESIDENT programs must be at least quasi-reentrant.

REUSABLE
The program is at least quasi-reentrant; a single copy in storage can be used by several tasks in the system. A REUSABLE program becomes eligible for removal from storage as part of the normal dynamic program storage compression (DPSC) scheme when its use count reaches zero.

TRANSIENT
Similar to REUSABLE, except that the program is removed from storage immediately when its use count reaches zero. Specify this option only for less-frequently used programs, or for programs in systems that are critically short on storage.

PROGRAM_NAME(name8 | string | "string")
Specifies the name of the program to be defined.

name8
The name of a location where there is an 8-byte program name.

string
A string of characters, without intervening blanks, naming the program.

"string"
A string of characters within quotation marks. The string length is set to 8 by padding with blanks or by truncation.

PROGRAM_TYPE(PRIVATE|SHARED|TYPE_ANY)
Specifies where to load the program from.

PRIVATE
The program is in the DFHRPL or dynamic LIBRARY concatenation. A PRIVATE program need not be reentrant, and is given only limited protection from unauthorized overwriting. The degree of protection depends on the type of dynamic storage area (DSA) into which the program is loaded (see the EXECUTION_KEY option):

DSA
Protection from unauthorized overwriting

CDSA
Cannot be overwritten by USER tasks.

ECDSA
Cannot be overwritten by USER tasks.

ERDSA
Complete: cannot be overwritten by USER tasks or CICS tasks.

EUDSA
None.

RDSA
Complete: cannot be overwritten by USER tasks or CICS tasks.

UDSA
None.

SHARED
The program is located in the link pack area (LPA), is reentrant, and is protected.

TYPE_ANY
Either the copy in DFHRPL or dynamic LIBRARY concatenation, or the LPA copy of the program can be used, though preference is given to the LPA copy.

REQUIRED_AMODE(24|31|AMODE_ANY|64)
Specifies the addressing mode of the program. If, during subsequent ACQUIRE_PROGRAM processing, no copy of the program that meets the defined addressing requirement can be found, the ACQUIRE_PROGRAM call receives an EXCEPTION response and the REASON value PROGRAM_NOT_FOUND.

Note:

1. AMODE_ANY and AMODE 31 have identical meanings for this function.
2. You cannot use this option to override the link-edited addressing mode of the program.

REQUIRED_RMODE(24|RMODE_ANY)
Specifies the residency mode of the program. If, during subsequent ACQUIRE_PROGRAM processing, no copy of the program that meets the defined addressing requirement can be found, the ACQUIRE_PROGRAM call receives an EXCEPTION response and the REASON value PROGRAM_NOT_FOUND.

Note: You cannot use this option to override the link-edited residence mode of the program.

RESPONSE and REASON values for DEFINE_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	CATALOG_ERROR

RESPONSE	*REASON*
	CATALOG_NOT_OPERATIONAL
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

The DELETE_PROGRAM call

DELETE_PROGRAM removes the definition of a named program from the catalog and from the list of current programs. When this request executes successfully, subsequent ACQUIRE_PROGRAM requests fail with a REASON value of 'PROGRAM_NOT_DEFINED'.

DELETE_PROGRAM

```
DFHLDLDX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(DELETE_PROGRAM),
      PROGRAM_NAME(name8 | string | 'string' ),]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

PROGRAM_NAME(name8 | string | "string")
specifies the name of the program to be deleted.

name8
The name of a location containing an 8-byte program name.

string
A string of characters naming the program.

"string"
A string in quotation marks. The string length is set to 8 by padding with blanks or truncating.

RESPONSE and REASON values for DELETE_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	PROGRAM_NOT_DEFINED
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

The IDENTIFY_PROGRAM call

IDENTIFY_PROGRAM locates the program that is associated with an address. If the address is in a CICS-defined program, the call returns information about that program.

If the address is not associated with a loader domain CICS-defined program, the request fails with a REASON value of INSTANCE_NOT_FOUND.

IDENTIFY_PROGRAM

```
IDENTIFY_PROGRAM
DFHLDLDX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(IDENTIFY_PROGRAM),
      ADDRESS(name4 | (Rn)| *),]
      [OUT,
      [PROGRAM_NAME(name8 | *),]
      [PROGRAM_ATTRIBUTE(name1 | (Rn) | *),]
      [PROGRAM_LENGTH(name4 | (Rn) | *),]
      [LOAD_POINT(name4 | (Ra) | *),]
      [ENTRY_POINT(name4 | (Ra) | *),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

ADDRESS(name4 | (Rn) | *)
The storage address that is used to identify the program.

name4
The name of a 4-byte fullword where the storage address is stored.

(Rn)
A register that is set to the storage address.

PROGRAM_NAME(name8 | *)
Returns the name of the program that contains the storage address. The PROGRAM_NAME corresponds to the CICS-defined program name, not a CSECT.

name8
The name of a location to contain an 8-byte program name.

PROGRAM_ATTRIBUTE(name1 | (Rn) | *)
Returns the program attribute.

name1
The name of a 1-byte location to receive the program attribute.

(Rn)
A register in which the low-order byte receives the program attribute and the other bytes are set to zero. The register can have the values RELOAD, RESIDENT, REUSABLE, or TRANSIENT.

RELOAD
The program is not reusable, and therefore several copies of the program might be loaded. A copy is removed from storage when a RELEASE_PROGRAM call (for that copy) is issued.

RESIDENT
There is a single copy of the program that is not removed from storage unless it is deleted. RESIDENT programs must be at least quasi-reentrant. Any program of PROGRAM_TYPE SHARED has the RESIDENT attribute by default. The DELETE_PROGRAM call has no effect on this type of RESIDENT program.

REUSABLE
The program is similar to RESIDENT, except that if the program is not in use, CICS can remove it from storage to optimize storage use.

TRANSIENT
The program is similar to RESIDENT, except that the program is removed from storage as soon as its use count drops to zero.

PROGRAM_LENGTH(name4 | (Rn) | *)
Returns the length of the named program.

name4
The name of a 4-byte location to receive the length in bytes, expressed in binary.

(Rn)
A register to contain the length in bytes, expressed in binary.

LOAD_POINT(name4 | (Ra) | *)
Returns the load point address of the program.

name4
The name of a 4-byte location to receive the loaded address.

(Ra)
A register to contain the load address.

ENTRY_POINT(name4 | (Ra) | *)
Returns the entry point address of the program.

name4
The name of a 4-byte location to receive the 31-bit entry address.

(Ra)
A register to receive the entry address.

RESPONSE and REASON values for IDENTIFY_PROGRAM

RESPONSE	REASON
OK	None
EXCEPTION	INSTANCE_NOT_FOUND
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more information, refer to the explanation of RESPONSE and REASON in Making an XPI call.

The RELEASE_PROGRAM call

RELEASE_PROGRAM decrements the use count of a currently loaded program by one.

If the program has been defined with the RELOAD attribute, the storage occupied by this copy of the program is released.

You should issue the ACQUIRE_PROGRAM and RELEASE_PROGRAM requests for a single program during the same execution of the exit program. If you do not want to do this, you should acquire the program once during CICS initialization, and leave it resident until CICS termination.

RELEASE_PROGRAM

```
DFHLDLDX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(RELEASE_PROGRAM),
      ENTRY_POINT(pointer),
      {PROGRAM_NAME(name8 | string | 'string')|
      PROGRAM_TOKEN(name8)},]
```

```
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

ENTRY_POINT(pointer)
Specifies the address of the entry point of this copy of the named program.

PROGRAM_NAME(name8 | string | "string")
Specifies the name of the program to be released.

name8
The name of a location containing an 8-byte program name.

string
A string of characters naming the program.

"string"
A string in quotation marks. The string length is set to 8 by padding with blanks or truncating.

PROGRAM_TOKEN(name8),
Specifies a token identifying the program to be released.

name8
The name of a location containing the 8-byte token obtained from a previous DEFINE_PROGRAM or ACQUIRE_PROGRAM call.

RESPONSE and REASON values for RELEASE_PROGRAM

RESPONSE	***REASON***
OK	None
EXCEPTION	PROGRAM_NOT_DEFINED
	PROGRAM_NOT_IN_USE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note:

1. For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.
2. ‘PROGRAM_NOT_DEFINED’ is returned if the program that you name is not known to the system.
3. ‘PROGRAM_NOT_IN_USE’ is returned when the use count for the named program is already zero.

Chapter 8. Log manager XPI functions

The XPI provides two log manager functions. These are the DFHLGPAX calls INQUIRE_PARAMETERS and SET_PARAMETERS. You can use these calls to inquire upon, and set, the log manager parameter, KEYPOINT_FREQUENCY. This parameter specifies the activity keypoint frequency of the CICS region.

The INQUIRE_PARAMETERS call

INQUIRE_PARAMETERS returns information about the activity keypoint frequency of the system.

INQUIRE_PARAMETERS

```
DFHLGPAX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(INQUIRE_PARAMETERS),
      [OUT,
      [KEYPOINT_FREQUENCY(name4 | *),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

KEYPOINT_FREQUENCY(name4 | *)
returns the activity keypointing frequency of the CICS region.

name4
The name of a 4-byte location that is to receive the frequency value.

RESPONSE and REASON values for INQUIRE_PARAMETERS

RESPONSE	*REASON*
OK	None
DISASTER	None
INVALID	None
KERNERROR	None

The SET_PARAMETERS call

SET_PARAMETERS allows you to set the activity keypoint frequency for the CICS region.

SET_PARAMETERS

```
DFHLGPAX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(SET_PARAMETERS),
      [KEYPOINT_FREQUENCY(name4 | (Rn) ),]]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

KEYPOINT_FREQUENCY(name4 | *)
specifies the activity keypointing frequency of the CICS region.

Permitted values are 0, or any integer between 200 and 65535 inclusive.

name4
The name of a 4-byte location that contains the new frequency value.

(Rn)
A register that contains the new frequency value.

RESPONSE and REASON values for SET_PARAMETERS

RESPONSE	*REASON*
OK	None
EXCEPTION	OUT_OF_RANGE
DISASTER	None
INVALID	None
KERNERROR	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

Chapter 9. Monitoring XPI functions

The XPI provides four monitoring functions: the DFHMNMNX calls INQUIRE_MONITORING_DATA and MONITOR, the DFHMNTDX call SET_TRACKING_DATA, and the DFHMNIAX call INQUIRE_APP_CONTEXT.

Restriction:

DFHMNMNX, DFHMNTDX, and DFHMNIAX calls cannot be used in any exit program that is called from any global user exit point in the following domains:

- Dispatcher domain
- Dump domain
- Monitor domain
- Statistics domain
- Transient data program

In addition, the SET_TRACKING_DATA and INQUIRE_APP_CONTEXT calls cannot be used in any exit program that is called from any global user exit point in the following domains:

- Transaction manager domain

Also, the INQUIRE_APP_CONTEXT, INQUIRE_MONITORING_DATA, and SET_TRACKING_DATA calls cannot be used in any exit program that is called from any global user exit point in DFHTCP or DFHZCP.

The INQUIRE_APP_CONTEXT call

The INQUIRE_APP_CONTEXT call returns to the exit program the application context data that is associated with the current application running the issuing task.

Restriction

Do not start exit programs that use the INQUIRE_APP_CONTEXT function until the second phase of the PLTPI. For more information about the PLTPI, see Writing initialization and shutdown programs .

INQUIRE_APP_CONTEXT

```
DFHMNIAX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(INQUIRE_APP_CONTEXT),
      CONTEXT(INITIAL | CURRENT)]
      [OUT,
      [APPLNAME(name64),]
      [PLATNAME(name64),]
      [OPERNAME(name64),]
      [MAJORVER(name4 | (Rn)),]
      [MINORVER(name4 | (Rn)),]
      [MICROVER(name4 | (Rn)),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

APPLNAME(name64)
Returns the 64-byte name of the application that is associated with the task.

CONTEXT(INITIAL | CURRENT)
The parameter **CONTEXT(INITIAL | CURRENT)** determines whether the application context values returned are for a tasks initial context or its current context. If the **CONTEXT** parameter is not specified, the default is to return the tasks current application context.

MAJORVER(name4 | (Rn))
Returns the major version number of the application that is associated with the task.

name4
The name of a 4-byte field that contains the major version number as a binary value.

(Rn)
A register to receive the major version number.

MICROVER(name4 | (Rn))
Returns the micro version number of the application that is associated with the task.

name4
The name of a 4-byte field that contains the micro version number as a binary value.

(Rn)
A register to receive the micro version number.

MINORVER(name4 | (Rn))
Returns the minor version number of the application that is associated with the task.

name4
The name of a 4-byte field that contains the minor version number as a binary value.

(Rn)
A register to receive the minor version number.

OPERNAME(name64)
Returns the 64-byte name of the operation that is associated with the task.

PLATNAME(name64)
Returns the 64-byte name of the platform that is associated with the task.

RESPONSE and REASON values for INQUIRE_APP_CONTEXT

RESPONSE	REASON
OK	None
EXCEPTION	APP_CONTEXT_UNAVAILABLE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

For more information, see the explanation of RESPONSE and REASON in Making an XPI call.

The INQUIRE_MONITORING_DATA call

The INQUIRE_MONITORING_DATA function returns to the exit program the performance class monitoring data that has been accumulated for the issuing task.

The DFHMNTDS DSECT that maps the data is of fixed format. Note that:

- *All* the CICS system-defined fields in the performance records (including fields that you have specified for exclusion using the EXCLUDE option of the DFHMCT TYPE=RECORD macro) are listed.
- *No* user-defined data fields are listed.

INQUIRE_MONITORING_DATA

```
DFHMNMNX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(INQUIRE_MONITORING_DATA),
      DATA_BUFFER(buffer-descriptor),]
      [OUT,
```

```
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

Important

There is a restriction in using the XPI early during initialization. Do not start exit programs that use the XPI functions TRANSACTION_DUMP, WRITE_JOURNAL_DATA, MONITOR, and INQUIRE_MONITOR_DATA until the second phase of the PLTPI. For further information about the PLTPI, refer to Writing initialization and shutdown programs .

DATA_BUFFER(buffer-descriptor)
specifies the address and the length of a buffer to contain the returned monitoring data; see XPI syntax for a full definition of a buffer-descriptor. The DSECT DFHMNTDS maps the monitoring data.

RESPONSE and REASON values for INQUIRE_MONITORING_DATA

RESPONSE	*REASON*
OK	None
EXCEPTION	LENGTH_ERROR
	MONITOR_DATA_UNAVAILABLE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note:

1. For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.
2. 'LENGTH_ERROR' means that the length specified in the buffer-descriptor was too short for the monitoring data returned from the XPI call.

The MONITOR call

The MONITOR XPI call is similar to the **EXEC CICS MONITOR** command. It enables you to invoke user event-monitoring points (EMPs) in your exit programs.

The user event-monitoring points must be defined in the monitoring control table (MCT) using the DFHMCT TYPE=EMP macro, or generated by the **APPLNAME** parameter on the DFHMCT TYPE=INITIAL macro. For more information about CICS monitoring, see CICS monitoring facility: Performance and tuning .

At a user EMP, you can add your own data (up to 256 counters, up to 256 clocks, and a single character string of up to 256 bytes) to fields reserved unconditionally for you in performance class monitoring data records. You can also add up to 12 bytes of user information at the DFHAPPL EMPs.

MONITOR

```
DFHMNMNX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(MONITOR),
      POINT(expression | name2 | (Rn)),
      [DATA1(expression | name4 | (Ra) | *),]
      [DATA2(expression | name4 | (Ra) | *),]
      [ENTRYNAME(name8 | string | 'string'),]]
      [OUT,
```

```
       RESPONSE(name1 | *),
       REASON(name1 | *)]
```

This command is threadsafe.

Important

There is a restriction in using the XPI early during initialization. Do not start exit programs that use the XPI functions TRANSACTION_DUMP, WRITE_JOURNAL_DATA, MONITOR, and INQUIRE_MONITOR_DATA until the second phase of the PLTPI. For further information about the PLTPI, see Writing initialization and shutdown programs .

DATA1(expression | name4 | (Ra) | *)
Specifies a fullword binary variable whose contents depend on the type of user EMP being used:

- If the MCT user EMP definition contains an ADDCNT, SUBCNT, NACNT, EXCNT, or ORCNT option, the DATA1 variable is an area used as defined by the user EMP definition.
- If the MCT user EMP definition contains an MLTCNT option, the DATA1 variable is an area with the address of a series of adjacent fullwords containing the values to be added to the user count fields defined in the user EMP definition.
- If the MCT user EMP definition contains a MOVE option, the DATA1 variable is an area with the address of the character string to be moved. This rule also applies to the DFHAPPL EMPs.

For details of the user EMP options, see Monitoring control table (MCT).

expression
A valid assembler-language expression giving the fullword binary variable for this EMP.

name4
The name of a 4-byte field containing the fullword binary variable for this EMP.

(Ra)
A register containing the fullword binary variable for this EMP.

*****
The value of this option is already present in the parameter list, or the option is not specified for this EMP.

DATA2(expression | name4 | (Rn) | *)
Specifies a fullword binary variable whose contents depend on the type of user EMP being used:

- If the MCT user EMP definition contains an ADDCNT, SUBCNT, NACNT, EXCNT, or ORCNT option, the DATA2 variable is an area used as defined by the user EMP definition.
- If the MCT user EMP definition contains an MLTCNT option, the DATA2 variable is an area with the number of user count fields to be updated.

  The number specified in DATA2 overrides the default value defined in the MCT for the operation. A value of 0 instructs monitoring to use the default. Not specifying a value for DATA2 does not prevent the MLTCNT operation from being successful; but, if it is, an exception response of DATA2_NOT_SPECIFIED is returned. See note 5.
- If the MCT user EMP definition contains a MOVE option, the DATA2 variable is an area with the length of the character string to be moved.

  The length specified in DATA2 overrides the default value defined in the MCT for the operation. A value of 0 instructs monitoring to use the default. Not specifying a value for DATA2 does not prevent the MOVE operation from being successful; but, if it is, an exception response of DATA2_NOT_SPECIFIED is returned. See note 5.

For details of the user EMP options, see Monitoring control table (MCT).

expression
A valid assembler-language expression giving the fullword binary variable for this EMP.

name4
The name of a 4-byte field containing the fullword binary variable for this EMP.

(Rn)
A register containing the fullword binary variable for this EMP.

*****
The value of this option is already present in the parameter list, or the option is not specified for this EMP.

ENTRYNAME(name8 | string | "string")
Specifies the monitoring point entry name, which qualifies the POINT value and which is defined in the monitoring control table (MCT).

name8
The name of a location containing an 8-byte string.

string
A string of characters without intervening blanks. The macro generates, from the string, a literal constant of length 8 bytes, extending with blanks or truncating as required.

"string"
A string, enclosed in quotation marks, and possibly containing blanks. This value is processed in the same way as the previous “string”.

Note: If, when defining the EMP in the MCT, you do not specify an entry name, the entry name defaults to ‘USER'. ENTRYNAME likewise defaults to ‘USER' if not specified.

POINT(expression | name2 | (Rn))
Specifies the monitoring point identifier as defined in the MCT, and is in the range 0 through 255. Point identifiers in the range 200 through 255 are reserved for use by IBM program products.

expression
A valid assembler-language expression that can be expressed in 2 bytes.

name2
The name of a 2-byte source of point data

(Rn)
A register containing the point data in the low-order 2 bytes

RESPONSE and REASON values for MONITOR

RESPONSE	REASON
OK	None
EXCEPTION	DATA1_NOT_SPECIFIED
	DATA2_NOT_SPECIFIED
	POINT_NOT_DEFINED
	INVALID_DATA1_VALUE
	INVALID_DATA2_VALUE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note:

1. For more detail, see the explanation of RESPONSE and REASON in Making an XPI call.
2. POINT_NOT_DEFINED means that the EMP you have specified was not defined in the MCT.
3. INVALID_DATA1_VALUE and INVALID_DATA2_VALUE are most likely to have been caused by provision of bad addresses; this causes a program check.

4. DATA1_NOT_SPECIFIED and DATA2_NOT_SPECIFIED mean that you have not specified DATA1 or DATA2 respectively when the operation required them. See the description of DATA2.
5. Any error response terminates processing of the EMP. Operations defined to execute before the point of failure will have done so; later operations are canceled.

The SET_TRACKING_DATA call

The SET_TRACKING_DATA call function sets the transaction tracking origin data tag for the issuing task.

SET_TRACKING_DATA

```
DFHMNTDX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(SET_TRACKING_DATA),
      {TRACKING_TAG (MOBILE)|TRACKING_TAG_VALUE(name1 | *)},]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

Important

There is a restriction in using the XPI early during initialization. Do not start exit programs that use the SET_TRACKING_DATA function until the second phase of the PLTPI. For more information about the PLTPI, see Writing initialization and shutdown programs .

TRACKING_TAG (MOBILE)

Specifies the transaction tracking origin data tag information to be set in the transaction tracking origin data.

MOBILE
The transaction tracking origin data tag is mobile.

RESPONSE and REASON values for SET_TRACKING_DATA

RESPONSE	REASON
OK	None
EXCEPTION	NO_ASSOCIATION_DATA
	TRACKING_TAG_ALREADY_SET
	INVALID_TRACKING_TAG
	INVALID_TRACKING_TAG_VALUE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note:

1. For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.
2. NO_ASSOCIATION_DATA means that the transaction under which this XPI call has been invoked has no transaction tracking association data. For more information about association data and transaction tracking, see Introduction to CICS intercommunication.

3. TRACKING_TAG_ALREADY_SET means that transaction tracking origin data tag for the issuing task has already been set.
4. INVALID_TRACKING_TAG means that the transaction tracking origin data tag has an invalid value.
5. INVALID_TRACKING_TAG_VALUE means that the transaction tracking origin data tag value is not in the range of 129 through to 255.

Chapter 10. Object transaction XPI functions

You can use the object transaction XPI calls to implement a TRUE program that responds to calls between a CICS unit of work and a remote transaction coordinator. These are the DFHOTTRX calls COMMIT, COMMIT_ONE_PHASE, IMPORT_TRAN, PREPARE, ROLLBACK, and SET_ROLLBACK_ONLY, and the DFHOTCOX call SET_COORDINATOR. These functions provide powerful control over the sync point processing of a CICS unit of work. If you use these calls incorrectly, the CICS Recovery Manager terminates CICS immediately and recovery and resynchronization processing is required.

The IMPORT_TRAN call

Links the current unit of work of a task to an external transaction. Some information about the external transaction is recorded in the current unit of work.

IMPORT_TRAN

```
DFHOTTRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(IMPORT_TRAN),
       [FORMAT_ID,(name4|Rn),]
       [BQUAL_LEN,(name4|Rn),]
       [TID_BLOCK_IN,(block-descriptor),]
       [TIMEOUT,(name4|Rn),]
       [LOGICAL_SERVER,(name4|string|'string'),]
       [PUBLIC_ID,(name64|string|'string'),]]
       [OUT,
       UOW_ID,(name8 | *),
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

BQUAL_LEN (name4 | Rn)
Specifies the branch qualifier length of the OTS transaction identifier (TID).

name4
The name of a 4-byte location containing the branch qualifier length.

Rn
A register containing the branch qualifier length.

FORMAT_ID (name4 | Rn)
Specifies the OTS transactions format identifier.

name4
The name of a 4-byte location containing the format ID.

Rn
A register containing the format ID.

LOGICAL_SERVER (name4 | string | 'string')
Specifies the name of the logical server in which the transaction is executing. Users of this XPI should choose values that are not similar to any CorbaServer definition that is installed in the CICS system.

PUBLIC_ID (name64 | string | 'string')
Specifies the public identifier associated with the transaction.

TID_BLOCK_IN (block-descriptor)
Specifies the unique OTS transaction identifier (TID) of the external transaction that will be associated with the task's unit of work. The block-descriptor is two fullwords of data, where the first word contains the address of the data, and the second word contains the length in bytes of the data.

TIMEOUT (name4 | Rn)
Specifies the OTS transaction timeout value in seconds.

name4
The name of a 4-byte location containing the timeout value.

Rn
A register containing the timeout value.

UOW_ID (name8 | *)
Specifies the identifier of the CICS unit of work into which the OTS transaction was imported.

RESPONSE and REASON values for IMPORT_TRAN

RESPONSE	REASON
OK	None
EXCEPTION	TID_TOO_LONG OTS_TRAN_ALREADY
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The COMMIT_ONE_PHASE call

Performs a sync point on the unit of work of the current task, without referencing an external coordinator. You cannot use the COMMIT_ONE_PHASE call if you have used the SET_COORDINATOR call to add information about a coordinator to the unit of work of the current task.

COMMIT_ONE_PHASE

```
DFHOTTRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(COMMIT_ONE_PHASE),]]
       [OUT,
       STATUS (name1 | *),
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

STATUS (name1 | *)
The outcome of the CICS unit of work.

This parameter can have the following values:

COMMITTED
ROLLEDBACK

RESPONSE and REASON values for COMMIT_ONE_PHASE

RESPONSE	REASON
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The PREPARE call

Performs the first phase of the sync point on the CICS unit of work on behalf of an OTS transaction. The vote returned by this function is intended to be used by the coordinator of the OTS transaction to determine the outcome of the overall transaction. The CICS unit of work enters the *indoubt* state when PREPARE is called and remains indoubt until a subsequent COMMIT or ROLLBACK function is called. If the CICS system fails and needs to be restarted, the CICS unit of work is recovered from the system log and you must resynchronize to resolve the indoubt unit of work.

PREPARE

```
DFHOTTRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(PREPARE),]
       [OUT,
       VOTE (name1 | *),
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

VOTE (name1 | *)
The outcome of the first phase of the OTS transaction.

This parameter can have the following values:

YES
NO
READ_ONLY
HEURISTIC_MIXED

RESPONSE and REASON values for PREPARE

RESPONSE	REASON
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The COMMIT call

Performs the second phase of the sync point of an OTS transaction, ensuring that the transaction is committed.

COMMIT

```
DFHOTTRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(COMMIT),]
       [OUT,
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

RESPONSE and REASON values for COMMIT

RESPONSE	REASON
OK	None
EXCEPTION	UOW_ROLLEDBACK
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The ROLLBACK call

Rolls back an OTS transaction.

ROLLBACK

```
DFHOTTRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(ROLLBACK),]
       [OUT,
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

RESPONSE and REASON values for ROLLBACK

RESPONSE	REASON
OK	None
EXCEPTION	UOW_COMMITTED
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The SET_ROLLBACK_ONLY call

Marks the CICS unit of work so that the OTS coordinator is given a 'NO' vote when a sync point protocol is performed, forcing the rollback of the global transaction. The CICS unit of work continues in the *inflight* state, but any subsequent recoverable resource updates are rolled back with the rest of the global transaction.

SET_ROLLBACK_ONLY

```
DFHOTTRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(SET_ROLLBACK_ONLY),]
       [OUT,
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

RESPONSE and REASON values for SET_ROLLBACK_ONLY

RESPONSE	REASON
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The SET_COORDINATOR call

Associates a link with the current task's unit of work to represent a remote coordinator.

SET_COORDINATOR

Ensure that the object transaction XPI functions for phase 1 (OTTR_PREPARE) and phase 2 (OTTR_COMMIT or OTTR_ROLLBACK) are used to drive the current unit of work through sync point processing, in response to the actions of the remote coordinator. You cannot use the OTTR_COMMIT_ONE_PHASE function if you have used the SET_COORDINATOR function to add information about a coordinator to the unit of work of the current task, and you cannot allow the task to end without using the correct object transaction XPIs to sync point the current unit of work.

```
DFHOTCOX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(SET_COORDINATOR),
       [IOR_BLOCK,(block-descriptor)]
       [HOST_BLOCK,(block-descriptor)]]
       [OUT,
       COORDINATOR_TOKEN,(name1 | *),
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

HOST_BLOCK (block-descriptor)
A pointer to and length of a character string that represents the identity of the system, which contains the coordinator instance. The maximum supported length of this parameter is 4096 bytes.

IOR_BLOCK (block-descriptor)
A pointer to and length of a character string that represents the coordinator instance in the host system. The maximum supported length of this parameter is 4096 bytes.

COORDINATOR_TOKEN (name1 | *)
A token representing the coordinator.

RESPONSE and REASON values for SET_COORDINATOR

RESPONSE	REASON
OK	None
EXCEPTION	IOR_TOO LONG HOST_TOO_LONG LINK_UNKNOWN COORDINATOR_NOT_FOUND COORDINATOR_ALREADY INVALID_SYNCPOINT_STATE

RESPONSE	REASON
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Chapter 11. Parameter domain XPI functions

The XPI provides one parameter domain function, the DFHPAIQX call INQUIRE_FEATUREKEY for feature toggles.

The INQUIRE_FEATUREKEY call

INQUIRE_FEATUREKEY obtains the value for a feature toggle.

For a list of toggle-enabled features by CICS release, see Toggle-enabled features, support by release. Follow the links in the feature list table to locate the information that describes the feature toggles for enabling and setting configuration options for a specific toggle-enabled feature.

INQUIRE_FEATUREKEY

```
DFHPAIQX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(INQUIRE_FEATUREKEY),
      OPTION(name255|'string'),
      [STRING(buffer-descriptor),]
      [UPPERCASE (YES|NO),]
      [OUT,
      [NUMBER(name4),]
      [BOOLEAN(name1)]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

OPTION(name255|'string')
Specifies the name of the feature toggle.

STRING(buffer-descriptor)
Specifies the address and the length of a buffer to contain the returned character string value of the toggle. See XPI syntax for a full definition of a buffer-descriptor.

UPPERCASE (YES|NO)
Specifies if the value returned in STRING is changed to uppercase.

NUMBER(name4)
Returns a numeric toggle value in binary.

BOOLEAN(name1)
Returns a boolean toggle value in boolean.

TRUE
The value is true.

FALSE
The value is false.

RESPONSE and REASON values for INQUIRE

RESPONSE	*REASON*
OK	None

RESPONSE	***REASON***
EXCEPTION	NOT_FOUND
	BUFFER_TOO_SMALL
	BAD_OPTION
	NOT_BOOLEAN
	NOT_NUMBER
	NO_LIST
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Chapter 12. Program management XPI functions

The XPI provides eight program management functions, including DFHPGISX, DFHPGAQX, and DFHPGCHX calls. Together with the loader functions, these functions provide a comprehensive set of tools to manipulate programs.

The program management functions include the following DFHPGISX calls:

- END_BROWSE_PROGRAM
- GET_NEXT_PROGRAM
- INQUIRE_CURRENT_PROGRAM
- INQUIRE_PROGRAM
- SET_PROGRAM
- START_BROWSE_PROGRAM

The program management functions include the following DFHPGAQX calls:

- INQUIRE_AUTOINSTALL
- SET_AUTOINSTALL

The program management functions include the following DFHPGCHX call:

- BIND_CHANNEL

You can use these functions together with the loader functions (DFHLDLDX calls) to manipulate programs. However, the tokens returned in the NEW_PROGRAM_TOKEN fields of DFHPGISX calls are different from the tokens that are returned by DFHLDLDX calls. Do not use a token obtained from a DFHPGISX call in a DFHLDLDX call, or vice versa.

The INQUIRE_PROGRAM call

INQUIRE_PROGRAM returns information about the attributes of a specified program.

INQUIRE_PROGRAM

```
DFHPGISX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(INQUIRE_PROGRAM),
      [AC_APPLICATION_NAME(block-descriptor),]
      [AC_MAJOR_VERSION(name4),]
      [AC_MICRO_VERSION(name4),]
      [AC_MINOR_VERSION(name4),]
      [AC_PLATFORM_NAME(block-descriptor),]
      [{PROGRAM_NAME(name8 | string | 'string')|
      PROGRAM_TOKEN(name4)},]
      [SHOW_PROGRAMS(PRIVATE|PRIVATE_AND_PUBLIC),]
      [OUT,
      [ACCESS(CICS|NONE|READ_ONLY|USER),]
      [APIST(CICSAPI|OPENAPI),]
      [AVAIL_STATUS(DISABLED|ENABLED),]
      [CEDF_STATUS(CEDF|NOCEDF|NOT_APPLIC),]
      [CONCURRENCY(QUASIRENT|THREADSAFE),]
      [DATA_LOCATION(ANY|BELOW|NOT_APPLIC),]
      [DYNAMIC_STATUS(DYNAMIC|NOT_DYNAMIC),]
      [ENTRY_POINT(name4),]
      [EXECUTION_KEY(CICS|NOT_APPLIC|USER),]
      [EXECUTION_SET(DPLSUBSET|FULLAPI|NOT_APPLIC),]
      [HOLD_STATUS(CICS_LIFE|NOT_APPLIC|TASK_LIFE),]
      [INSTALL_TYPE(AUTO|CATALOG|GROUPLIST|MANUAL|RDO|SYSAUTO),]
      [LANGUAGE_DEDUCED(ASSEMBLER|C370|COBOL|
                        COBOL2|LE370|NOT_APPLIC|NOT_DEDUCED|PLI),]
      [LANGUAGE_DEFINED(ASSEMBLER|C370|COBOL|
                        LE370|NOT_APPLIC|NOT_DEFINED|PLI),]
      [LIBRARY(name8),]
      [LIBRARYDSN(name44),]
```

```
      [LOAD_POINT(name4),]
      [LOAD_STATUS(LOADABLE|NOT_APPLIC|NOT_LOADABLE|NOT_LOADED),]
      [LOCATION(CDSA|ECDSA|ELPA|ERDSA|ESDSA|LPA|NONE|RDSA|SDSA),]
      [MODULE_TYPE(MAPSET|PARTITIONSET|PROGRAM),]
      [NEW_PROGRAM_TOKEN(name4),]
      [PROGRAM_ATTRIBUTE(RELOAD|RESIDENT|REUSABLE|TRANSIENT),]
      [PROGRAM_LENGTH(name4),]
      [PROGRAM_TYPE(NOT_APPLIC|PRIVATE|SHARED|TYPE_ANY),]
      [PROGRAM_USAGE(APPLICATION|NUCLEUS),]
      [PROGRAM_USE_COUNT(name4),]
      [PROGRAM_USER_COUNT(name4),]
      [REMOTE_DEFINITION(LOCAL|REMOTE),]
      [REMOTE_PROGID(name8),]
      [REMOTE_SYSID(name4),]
      [REMOTE_TRANID(name4),]
      [SPECIFIED_AMODE(24|31|AMODE_ANY|AMODE_NOT_SPECIFIED|64),]
      [SPECIFIED_RMODE(24|RMODE_ANY|RMODE_NOT_SPECIFIED),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

AC_APPLICATION_NAME(block-descriptor)
Specifies the address and length of the name of the application associated with the program. To inquire on private programs for applications deployed on platforms, you must specify the AC_APPLICATION_NAME, AC_MAJOR_VERSION, AC_MINOR_VERSION, AC_MICRO_VERSION, and AC_PLATFORM_NAME fields to provide a complete application context. For more information on block-descriptors, see XPI syntax.

AC_MAJOR_VERSION(name4)
Specifies the application major version in binary.

AC_MICRO_VERSION(name4)
Specifies the application micro version in binary.

AC_MINOR_VERSION(name4)
Specifies the application minor version in binary.

AC_PLATFORM_NAME(block-descriptor)
Specifies the address and length of the name of the platform associated with the program. For more information on block-descriptors, see XPI syntax.

ACCESS(CICS|NONE|READ_ONLY|USER)
Returns a value that indicates the type of storage into which the program has been loaded.

CICS
CICS-key.

NONE
The program has not been loaded.

READ_ONLY
Read-only.

USER
User-key.

APIST(CICSAPI|OPENAPI)
Returns a value that indicates the API attribute of the installed program definition.

CICSAPI
The program is restricted to use of only the CICS permitted application programming interfaces.

OPENAPI
The program is not restricted to use of only the CICS permitted application programming interfaces. The program must be coded to threadsafe standards and defined with CONCURRENCY(THREADSAFE).

AVAIL_STATUS(DISABLED|ENABLED)
Returns a value that indicates whether the program can be used (ENABLED) or not (DISABLED).

CEDF_STATUS(CEDF|NOCEDF|NOT_APPLIC)
Returns the execution diagnostic facility (EDF) status of the program.

CEDF
When the program is running under the control of the CICS EDF, EDF diagnostic screens are displayed.

NOCEDF
EDF diagnostic screens are not displayed.

NOT_APPLIC
Not applicable. This module is a mapset, a partitionset, or a remote program.

CONCURRENCY(QUASIRENT|THREADSAFE)
Returns a value that indicates the concurrency attribute of the installed program definition.

QUASIRENT
The program is defined as being quasi-reentrant, and can run only under the CICS QR TCB.

THREADSAFE
The program is defined as threadsafe, and can run under whichever TCB is in use by its user task when the program is given control. This could be either an open TCB or the CICS QR TCB.

Note: For a Language Environment-conforming program, the concurrency as originally defined can be overridden when the program is subsequently loaded.

DATA_LOCATION(ANY|BELOW|NOT_APPLIC)
Returns a value that indicates whether the program can access data located above the 16 MB line.

ANY
The program can handle 31-bit addresses, and can be passed data that is located above or below the 16 MB line.

BELOW
The program can handle only 24-bit addresses, and must therefore be passed only data that is located below the 16 MB line.

NOT_APPLIC
Not applicable. This module is a mapset, a partitionset, or a remote program.

DYNAMIC_STATUS(DYNAMIC|NOT_DYNAMIC)
Returns a value that indicates whether, if the program is the subject of a program-link request, the request can be dynamically routed.

DYNAMIC
If the program is the subject of a program-link request, the CICS dynamic routing program is invoked. Providing that a remote server region is not named explicitly on the SYSID option of the EXEC CICS LINK command, the routing program can route the request to the region on which the program is to run.

NOT_DYNAMIC
If the program is the subject of a program-link request, the dynamic routing program is not invoked.

For a distributed program link (DPL) request, the server region on which the program is to run must be specified explicitly on the REMOTESYSTEM option of the PROGRAM definition, or on the SYSID option of the EXEC CICS LINK command. Otherwise, it defaults to the local region.

For information about the dynamic routing of DPL requests, see Dynamically routing DPL requests.

ENTRY_POINT(name4)
Returns the entry point address of the program entry point address, as returned by a loader domain ACQUIRE_PROGRAM call.

EXECUTION_KEY(CICS|NOT_APPLIC|USER)
Returns the key in which CICS gives control to the program, which determines whether the program can modify CICS-key storage.

CICS
CICS gives control to the program in CICS key. The program is loaded into a CICS dynamic storage area (DSA) above or below the 16 MB line; that is, the CDSA or ECDSA, depending on its residency mode (RMODE) attribute, as defined to the linkage-editor.

NOT_APPLIC
Not applicable. This module is a mapset, a partitionset, or a remote program.

USER
CICS gives control to the program in user key. The program is loaded into a user DSA above or below the 16 MB line; that is, the UDSA or EUDSA, depending on its residency mode (RMODE) attribute, as defined to the linkage-editor.

EXECUTION_SET(DPLSUBSET|FULLAPI|NOT_APPLIC)
Returns a value that indicates whether CICS links to and runs the program as if it were running in a remote CICS region.

DPLSUBSET
CICS links to and runs the program with the API restrictions of a remote DPL program. The program can use only a subset of the CICS API.

FULLAPI
CICS links to and runs the program without the API restrictions of a remote DPL program. The program can use the full CICS API.

NOT_APPLIC
Not applicable. This module is a mapset, a partitionset, or a remote program. (The EXECUTIONSET option of DEFINE PROGRAM applies only to local program definitions. Its purpose is to test programs in a local CICS environment as if they were running as DPL programs.)

HOLD_STATUS(CICS_LIFE|NOT_APPLIC|TASK_LIFE)
Returns a value that indicates how long the program is to remain loaded.

CICS_LIFE
The program remains loaded until CICS is shut down.

NOT_APPLIC
Not applicable. The program is not loaded, or is remote.

TASK_LIFE
The program remains loaded for the lifetime of the task.

INSTALL_TYPE(AUTO|CATALOG|GROUPLIST|MANUAL|RDO|SYSAUTO)
Returns the method that was used to install the PROGRAM resource definition.

AUTO
Autoinstall.

CATALOG
The CICS global catalog, after a restart.

GROUPLIST
The CICS startup grouplist.

MANUAL
The program is a CICS internal module explicitly defined to the Program Manager by another CICS component.

RDO
RDO commands.

SYSAUTO
System autoinstall (that is, autoinstalled by CICS without calling the autoinstall user program). The program might be a CICS internal module or, for example, a first phase PLTPI program.

LANGUAGE_DEDUCED(ASSEMBLER|C370|COBOL|COBOL2|LE370| NOT_APPLIC|NOT_DEDUCED| PLI)
Returns the language deduced by CICS for the program. COBOL is OS/VS COBOL, which cannot run under this CICS version, and COBOL2 is either Enterprise COBOL or VS COBOL II.

LANGUAGE_DEFINED(ASSEMBLER|C370|COBOL|LE370| NOT_APPLIC|NOT_DEFINED|PLI)
Returns the programming language specified on the resource definition.

LIBRARY(name)
Returns the 8-character name of the LIBRARY resource from which this program was loaded. This field is blank if the program has not been loaded, or if the LPASTATUS is LPA (indicating that the program has been loaded from the LPA).

LIBRARYDSN(name44)
Returns the 44-character name of the data set from which the program was loaded. This field is blank if the program has not been loaded, or if the LPASTATUS is LPA (indicating that the program has been loaded from the LPA).

- If the program was loaded from an installed LIBRARY, the LIBRARY and LIBRARYDSN names will be returned.
- If the program was loaded from a LIBRARY that has been disabled, the LIBRARY name will be returned but the LIBRARYDSN will be blank.
- If the program was loaded from a LIBRARY that has been discarded, both LIBRARY and LIBRARYDSN will be blank.

LOAD_POINT(name4)
Returns the load point address of the program, as returned by a loader domain ACQUIRE_PROGRAM call.

LOAD_STATUS(LOADABLE|NOT_APPLIC|NOT_LOADABLE|NOT_LOADED)
Returns a value that indicates whether or not the program can be loaded.

LOADABLE
The program is loadable.

NOT_APPLIC
Not applicable. The program is remote.

NOT_LOADABLE
CICS has tried to load the program and failed; the program is not in the library.

NOT_LOADED
CICS has not yet tried to load the program.

LOCATION(CDSA|ECDSA|ELPA|ERDSA|ESDSA|LPA|NONE|RDSA|SDSA)
Returns a value that indicates where the most recently loaded copy of the program resides.

CDSA
The CICS dynamic storage area

ECDSA
The extended CICS dynamic storage area

ELPA
The extended link pack area

ERDSA
The extended read-only dynamic storage area

ESDSA
The extended shared dynamic storage area

LPA
The link pack area

NONE
The program has not been loaded.

RDSA
The read-only dynamic storage area

SDSA
The shared dynamic storage area

MODULE_TYPE(MAPSET|PARTITIONSET|PROGRAM)
Returns the type of program resource.

NEW_PROGRAM_TOKEN(name4)
Returns a token that can be used to identify the named program.

name4
The name of a location to receive a 4-byte token that identifies this program.

If PROGRAM_NAME is specified on the request, NEW_PROGRAM_TOKEN is set to a program token that can be used on subsequent requests for the same program. If PROGRAM_TOKEN is specified on the request, NEW_PROGRAM_TOKEN is set to the same value.

PROGRAM_ATTRIBUTE(RELOAD|RESIDENT|REUSABLE|TRANSIENT)
Returns the residency status of the program; that is, when its storage is released.

RELOAD
The program is not reusable, and therefore several copies might be loaded. A copy is removed from storage when a RELEASE_PROGRAM call (for that copy) is issued.

RESIDENT
There is a single copy of the program that is not removed from storage unless deleted. RESIDENT programs must be at least quasi-reentrant. Any program of PROGRAM_TYPE SHARED is RESIDENT by default.

REUSABLE
Similar to RESIDENT, except that CICS can remove a REUSABLE program that is not in use from storage, for storage optimization reasons.

TRANSIENT
Similar to RESIDENT, except that a TRANSIENT program is removed from storage as soon as its user count drops to zero.

PROGRAM_LENGTH(name4)
Returns the length of the program, in bytes, expressed in binary.

PROGRAM_NAME(name8 | string | 'string')
Specifies the name of the program to be queried.

name8
The name of a location that contains an 8-byte program name.

string
A string of characters that name the program.

'string'
A string of characters in quotation marks. The string length is set to 8 by padding with blanks or truncating.

PROGRAM_TOKEN(name4)
Specifies a token that identifies the program to be queried.

name4
The name of a location that contains a 4-byte token that was obtained from a previous INQUIRE_PROGRAM call.

PROGRAM_TYPE(NOT_APPLIC|PRIVATE|SHARED|TYPE_ANY)
Returns a value that indicates where the next new copy of the program is to be loaded from.

NOT_APPLIC
Not applicable. The program is remote.

PRIVATE
The program is to be loaded from the DFHRPL or dynamic LIBRARY concatenation.. A PRIVATE program need not be reentrant, and is given only limited protection against unauthorized overwriting. The degree of protection depends on the type of dynamic storage area into which the program is loaded (see the description of the PROGRAM_TYPE option of the DEFINE_PROGRAM call).

SHARED
The program is to be loaded from the link pack area (LPA). SHARED programs must be reentrant, and are protected.

The next time a NEWCOPY or PHASEIN is received, an LPA copy of the program is used if it is available. If no LPA version is available, the program is loaded from DFHRPL or dynamic LIBRARY concatenation.

TYPE_ANY
Either the copy in DFHRPL or a dynamic LIBRARY concatenation, or the LPA copy of the program can be used, although preference is given to the LPA copy.

PROGRAM_USAGE(APPLICATION|NUCLEUS)
Returns a value that indicates whether the program is used as a CICS nucleus program, or as a user application program.

PROGRAM_USE_COUNT(name4)
Returns the number of different users that have invoked the program.

PROGRAM_USER_COUNT(name4)
Returns the current number of users of the program.

REMOTE_DEFINITION(LOCAL|REMOTE)
Returns a value that indicates whether this program is a local or a remote resource. If it is a remote resource, CICS treats requests to link to the program as distributed program link (DPL) requests, and ships them to the remote region.

REMOTE_PROGID(name8)
Returns the name by which the program is known in the remote CICS region, if the program is a remote resource. If REMOTESYSTEM was specified on the PROGRAM definition, and REMOTENAME omitted, the remote name will be the same as the local name (that is, REMOTE_PROGID will default to the value of PROGRAM_NAME).

REMOTE_SYSID(name4)
Returns the name of the remote CICS region that owns the program, if the program is a remote resource.

REMOTE_TRANID(name4)
Returns the name of the transaction that the remote CICS attaches, and under which it runs the program, if the program is a remote resource.

SHOW_PROGRAMS(PRIVATE|PRIVATE_AND_PUBLIC)
If application context fields are specified for INQUIRE_PROGRAM, SHOW_PROGRAMS defines the scope of the search.

PRIVATE
Searches only for private programs.

PRIVATE_AND_PUBLIC
Searches for private programs first, then public programs.

SPECIFIED_AMODE(24|31|AMODE_ANY|AMODE_NOT_SPECIFIED|64)
Returns the addressing mode that was specified on a DEFINE_PROGRAM call.

SPECIFIED_RMODE(24|RMODE_ANY|RMODE_NOT_SPECIFIED)
Returns the residency mode (that is, whether the program should be loaded above or below the 16 MB line) that was specified on a DEFINE_PROGRAM call.

RESPONSE and REASON values for INQUIRE_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	PROGRAM_NOT_DEFINED_TO_LD
	PROGRAM_NOT_DEFINED_TO_PG

RESPONSE	*REASON*
	APP_CONTEXT_NOT_FOUND
DISASTER	ABEND
	LOCK_ERROR
INVALID	INVALID_PROGRAM_TOKEN
KERNERROR	None
PURGED	None

The INQUIRE_CURRENT_PROGRAM call

INQUIRE_CURRENT_PROGRAM returns information about the attributes of the program that is currently running. If this call is issued from within a global or task-related user exit, it returns the attributes of the global or task-related user exit program itself.

INQUIRE_CURRENT_PROGRAM

```
DFHPGISX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(INQUIRE_CURRENT_PROGRAM),]
      [IGNORE_EXITS(YES|NO),]
      [OUT,
      [AVAIL_STATUS(DISABLED|ENABLED),]
      [CEDF_STATUS(CEDF|NOCEDF|NOT_APPLIC),]
      [CURRENT_AMODE(24|31|64),]
      [CURRENT_CEDF_STATUS(CEDF|NOCEDF),]
      [CURRENT_ENTRY_POINT(name4),]
      [CURRENT_ENVIRONMENT(EXEC|GLUE|PLT|SYSTEM|TRUE|URM),]
      [CURRENT_EXECUTION_SET(DPLSUBSET|FULLAPI),]
      [CURRENT_LOAD_POINT(name4),]
      [CURRENT_PROGRAM_LENGTH(name4),]
      [CURRENT_PROGRAM_NAME(name8),]
      [DATA_LOCATION(ANY|BELOW|NOT_APPLIC),]
      [DYNAMIC_STATUS(DYNAMIC|NOT_DYNAMIC),]
      [EXECUTION_KEY(CICS|NOT_APPLIC|USER),]
      [EXECUTION_SET(DPLSUBSET|FULLAPI|NOT_APPLIC),]
      [HOLD_STATUS(CICS_LIFE|NOT_APPLIC|TASK_LIFE),]
      [IGNORE_EXITS(YES|NO),]
      [INSTALL_TYPE(AUTO|CATALOG|GROUPLIST|MANUAL|RDO|SYSAUTO),]
      [INVOKING_ENVIRONMENT (EXEC|GLUE|PLT|SYSTEM|TRUE|URM),]
      [INVOKING_PROGRAM_NAME(name8),]
      [LANGUAGE_DEDUCED(ASSEMBLER|C370|COBOL|
                        COBOL2|LE370|NOT_APPLIC|NOT_DEDUCED|PLI),]
      [LANGUAGE_DEFINED(ASSEMBLER|C370|COBOL|
                        LE370|NOT_APPLIC|NOT_DEFINED|PLI),]
      [LIBRARY(name8),]
      [LIBRARYDSN(name44),]
      [LOAD_STATUS(LOADABLE|NOT_APPLIC|NOT_LOADABLE|NOT_LOADED),]
      [MODULE_TYPE(MAPSET|PARTITIONSET|PROGRAM),]
      [NEW_PROGRAM_TOKEN(name4),]
      [REMOTE_DEFINITION(LOCAL|REMOTE),]
      [REMOTE_PROGID(name8),]
      [REMOTE_SYSID(name4),]
      [REMOTE_TRANID(name4),]
      [RETURN_PROGRAM_NAME(name8),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

Note: The options not described in the following list are identical to the equivalent options of the INQUIRE_PROGRAM call. See “The INQUIRE_PROGRAM call” on page 67.

CURRENT_AMODE(24|31|64)
Returns the addressing mode that the running program is currently using.

CURRENT_CEDF_STATUS(CEDF|NOCEDF)
Returns the EDF status of the current instance of the program. The value returned is the same as for CEDF_STATUS, which is the EDF status specified on the program definition. See the CEDF_STATUS option of INQUIRE_PROGRAM.

CURRENT_ENTRY_POINT(name4)
Returns the entry point address of the current program.

CURRENT_ENVIRONMENT(EXEC|GLUE|PLT|SYSTEM|TRUE|URM)
Returns the environment in which the current program is running; that is, the type of program it is.

EXEC
User application program.

GLUE
Global user exit program.

PLT
Program list table program.

SYSTEM
CICS system code.

TRUE
Task-related user exit program.

URM
User-replaceable program.

CURRENT_EXECUTION_SET(DPLSUBSET|FULLAPI)
Returns the API execution set used by the current instance of the program. The value returned is the same as for EXECUTION_SET (which is the API execution set specified on the program definition) *unless* this is the first program in a transaction, when the value can be different. This is because the DPLSUBSET attribute applies only to linked-to programs. It is ignored for the first program in a transaction, because this cannot be the target of a DPL call. Therefore, for the first program in a transaction, if EXECUTION_SET returns DPLSUBSET, CURRENT_EXECUTION_SET nevertheless returns FULLAPI. See the EXECUTION_SET option of INQUIRE_PROGRAM.

CURRENT_LOAD_POINT(name4)
Returns the load point address of the current program.

CURRENT_PROGRAM_LENGTH(name4)
Returns the length of the current program, in bytes, expressed in binary.

CURRENT_PROGRAM_NAME(name8)
Returns the name of the program that is currently running.

IGNORE_EXITS(YES|NO)
Specifies whether global user exit programs and task-related user exit programs are ignored when returning information about the program that invoked the current program, and to which control will be returned. Your setting for this option affects the values returned by the INVOKING_ENVIRONMENT, INVOKING_PROGRAM_NAME, and RETURN_PROGRAM_NAME options. If YES is specified (the default), global user exit programs and task-related user exit programs are ignored for these options. If NO is specified, where a global user exit program or task-related user exit program is involved, the options return information about the exit program.

INVOKING_ENVIRONMENT (EXEC|GLUE|PLT|SYSTEM|TRUE|URM)
Returns the environment from which the current program was invoked; that is, the environment that corresponds to the program named in INVOKING_PROGRAM_NAME. The values are as described for CURRENT_ENVIRONMENT.

INVOKING_PROGRAM_NAME(name8)
Returns the name of the most recent program to invoke the current program. If IGNORE_EXITS(NO) is specified, this might be a global user exit program or task-related user exit program, if one was involved. If IGNORE_EXITS(YES) is specified, which is the default, this is the most recent program that was *not* a global user exit program or task-related user exit program.

LIBRARY(name)
Returns the 8-character name of the LIBRARY resource from which this program was loaded. This field is blank if the program has not been loaded, or if the LPASTATUS is LPA (indicating that the program has been loaded from the LPA). If the program was loaded from an installed LIBRARY, the LIBRARY and LIBRARYDSN names are returned.

LIBRARYDSN(*data-area*)
Returns the 44-character name of the data set from which the program was loaded. This fields is blank if the program has not been loaded, or if the LPASTATUS is LPA (indicating that the program has been loaded from the LPA). If the program was loaded from an installed LIBRARY, the LIBRARY and LIBRARYDSN names are returned.

RETURN_PROGRAM_NAME(name8)
Returns the name of the program to which control will be returned. If IGNORE_EXITS(NO) is specified, this might be a global user exit program or task-related user exit program. If IGNORE_EXITS(YES) is specified, which is the default, this is the program to which control will be returned after any intermediate global user exit programs or task-related user exit programs have completed.

RESPONSE and REASON values for INQUIRE_CURRENT_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	NO_CURRENT_PROGRAM
DISASTER	LOCK_ERROR
	ABEND
INVALID	None
KERNERROR	None
PURGED	None

The SET_PROGRAM call

Use SET_PROGRAM to set selected attributes in the definition of a specified program.

SET_PROGRAM

```
DFHPGISX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(SET_PROGRAM),
      {PROGRAM_NAME(name8 | string | 'string')|
      PROGRAM_TOKEN(name4)},]
      [AVAIL_STATUS(DISABLED|ENABLED),]
      [CEDF_STATUS(CEDF|NOCEDF),]
      [EXECUTION_KEY(CICS|USER),]
      [EXECUTION_SET(DPLSUBSET|FULLAPI),]
      [PROGRAM_ATTRIBUTE(RELOAD|RESIDENT|REUSABLE|TRANSIENT),]
      [PROGRAM_TYPE(PRIVATE|SHARED|TYPE_ANY),]
      [PROGRAM_USAGE(APPLICATION|NUCLEUS),]
      [REQUIRED_AMODE(24|31|AMODE_ANY|64),]
      [REQUIRED_RMODE(24|RMODE_ANY),]]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

AVAIL_STATUS(DISABLED|ENABLED)
Specifies whether the program can be used (ENABLED) or not (DISABLED).

CEDF_STATUS(CEDF|NOCEDF)
Specifies whether, when the program is running under the control of the CICS execution diagnostic facility (EDF), EDF diagnostic screens are displayed.

EXECUTION_KEY(CICS|USER)
Specifies the key in which CICS is to give control to the program. The key determines whether the program can modify CICS-key storage.

CICS
CICS gives control to the program in CICS key. The program is loaded into a CICS dynamic storage area (DSA) above or below the 16 MB line; that is, the CDSA or ECDSA, depending on its residency mode (RMODE) attribute, as defined to the linkage-editor.

USER
CICS gives control to the program in user key. The program is loaded into a user DSA above or below the 16 MB line; that is, the UDSA or EUDSA, depending on its residency mode (RMODE) attribute, as defined to the linkage-editor.

Note: If the program has been link-edited as reentrant with AMODE(31),RMODE(ANY), the EXECUTION_KEY option is ignored, and it is loaded into the extended read-only DSA (ERDSA). For details of the type of storage allocated for the ERDSA, see RENTPGM system initialization parameter.

EXECUTION_SET(DPLSUBSET|FULLAPI)
Specifies whether CICS is to link to and run the program as if it were running in a remote CICS region.

Note: EXECUTION_SET applies only to local program definitions. Its purpose is to test programs in a local CICS environment as if they were running as DPL programs.

DPLSUBSET
CICS links to and runs the program with the API restrictions of a remote DPL program. The program can use only a subset of the CICS API.

FULLAPI
CICS links to and runs the program without the API restrictions of a remote DPL program. The program can use the full CICS API.

PROGRAM_ATTRIBUTE(RELOAD|RESIDENT|REUSABLE|TRANSIENT)
Specifies the residency status of the program; that is, when its storage is to be released.

RELOAD
The program is not reusable, and therefore several copies might be loaded. A copy is removed from storage when a RELEASE_PROGRAM call (for that copy) is issued.

RESIDENT
At any one time there will be no more than a single copy of the program in storage, and this will not be removed unless deleted. RESIDENT programs must be at least quasi-reentrant. Any program of PROGRAM_TYPE SHARED is RESIDENT by default.

REUSABLE
Similar to RESIDENT, except that CICS can remove a REUSABLE program that is not in use from storage, for storage optimization reasons.

TRANSIENT
Similar to RESIDENT, except that a TRANSIENT program is removed from storage as soon as its user count drops to zero.

PROGRAM_NAME(name8 | string | 'string')
Specifies the name of the program whose attributes are to be changed.

name8
The name of a location that contains an 8-byte program name.

string
A string of characters that name the program.

'string'
A string of characters in quotation marks. The string length is set to 8 by padding with blanks or truncating.

PROGRAM_TOKEN(name4)
Specifies a token that identifies the program.

name4
The name of a location that contains a 4-byte token that was obtained from a previous INQUIRE_PROGRAM, INQUIRE_CURRENT_PROGRAM, START_BROWSE_PROGRAM, or GET_NEXT_PROGRAM call.

PROGRAM_TYPE(PRIVATE|SHARED|TYPE_ANY)
Specifies where the program is to be loaded from.

PRIVATE
The program is in the DFHRPL or dynamic LIBRARY concatenation. A PRIVATE program need not be reentrant, and is given only limited protection against unauthorized overwriting. The degree of protection depends on the type of dynamic storage area into which the program is loaded (see the description of the PROGRAM_TYPE option of the DEFINE_PROGRAM call).

SHARED
The program is located in the link pack area (LPA), is reentrant, and is protected.

TYPE_ANY
Either the copy in the DFHRPL or a dynamic LIBRARY concatenation, or the LPA copy of the program can be used, though preference is given to the LPA copy.

PROGRAM_USAGE(APPLICATION|NUCLEUS)
Specifies whether the program is used as a CICS nucleus program, or as a user application program.

REQUIRED_AMODE(24|31|AMODE_ANY|64)
Specifies the addressing mode of the program. If, during subsequent processing, no copy of the program that meets the defined addressing requirement can be found, an exception occurs.

Note:

1. AMODE_ANY and 31 have identical meanings for this function.
2. You cannot use this option to override the link-edited addressing mode of the program.

REQUIRED_RMODE(24|AMODE_ANY)
Specifies the residency mode of the program (that is, whether it is to be loaded above or below the 16MB line). If, during subsequent processing, no copy of the program that meets the defined residency requirement can be found, an exception occurs.

Note: You cannot use this option to override the link-edited residency mode of the program.

RESPONSE and REASON values for SET_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	CEDF_STATUS_NOT_FOR_MAPSET
	CEDF_STATUS_NOT_FOR_PTNSET
	CEDF_STATUS_NOT_FOR_REMOTE
	EXEC_KEY_NOT_FOR_MAPSET
	EXEC_KEY_NOT_FOR_PTNSET
	EXEC_KEY_NOT_FOR_REMOTE
	EXEC_SET_NOT_FOR_MAPSET
	EXEC_SET_NOT_FOR_PTNSET
	EXEC_SET_NOT_FOR_REMOTE
	INCOMPATIBLE_BUNDLE_SET

RESPONSE	*REASON*
	PROGRAM_NOT_DEFINED_TO_LD
	PROGRAM_NOT_DEFINED_TO_PG
DISASTER	ABEND
	CATALOG_ERROR
	CATALOG_NOT_OPERATIONAL
	LOCK_ERROR
INVALID	INVALID_MODE_COMBINATION
	INVALID_PROGRAM_NAME
	INVALID_PROGRAM_TOKEN
	INVALID_TYPE_ATTRIB_COMBIN
KERNERROR	None
PURGED	None

The START_BROWSE_PROGRAM call

START_BROWSE_PROGRAM returns a token that enables you to begin browsing through program definitions, optionally starting at the definition of a specified program.

START_BROWSE_PROGRAM

```
DFHPGISX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(START_BROWSE_PROGRAM),
      [PROGRAM_NAME(name8 | string | 'string'),]
      [OUT,
      BROWSE_TOKEN(name4)
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

BROWSE_TOKEN(name4)
returns a token to be used on a GET_NEXT_PROGRAM call, to initiate a sequential browse of program definitions.

name4
The name of a location to receive a 4-byte token.

PROGRAM_NAME(name8 | string | 'string')
specifies the name of the program whose definition you want to look at first. The browsing sequence is alphabetical. If there is no program with the specified name, CICS returns a token for the next definition in the alphabetic sequence. If you do not specify a program, CICS returns a token for the first definition.

name8
The name of a location containing an 8-byte program name.

string
A string of characters naming the program.

'string'
A string of characters in quotation marks. The string length is set to 8 by padding with blanks or truncating.

RESPONSE and REASON values for START_BROWSE_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	
DISASTER	ABEND
	INVALID_DIRECTORY
	LOCK_ERROR
INVALID	None
KERNERROR	None
PURGED	None

The GET_NEXT_PROGRAM call

Use GET_NEXT_PROGRAM to inquire on the next program definition during a browse sequence that is initiated by START_BROWSE_PROGRAM. The browsing sequence is alphabetical. The end of the alphabetical list of definitions is indicated by an END_LIST exception response.

GET_NEXT_PROGRAM

```
DFHPGISX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(GET_NEXT_PROGRAM),
      BROWSE_TOKEN(name4),]
      [OUT,
      PROGRAM_NAME(name8),
      [ACCESS(CICS|NONE|READ_ONLY|USER),]
      [AVAIL_STATUS(DISABLED|ENABLED),]
      [CEDF_STATUS(CEDF|NOCEDF|NOT_APPLIC),]
      [DATA_LOCATION(ANY|BELOW|NOT_APPLIC),]
      [ENTRY_POINT(name4),]
      [EXECUTION_KEY(CICS|NOT_APPLIC|USER),]
      [EXECUTION_SET(DPLSUBSET|FULLAPI|NOT_APPLIC),]
      [HOLD_STATUS(CICS_LIFE|NOT_APPLIC|TASK_LIFE),]
      [INSTALL_TYPE(AUTO|CATALOG|GROUPLIST|MANUAL|RDO|SYSAUTO),]
      [LANGUAGE_DEDUCED(ASSEMBLER|C370|COBOL|
                        COBOL2|LE370|NOT_APPLIC|NOT_DEDUCED|PLI),]
      [LANGUAGE_DEFINED(ASSEMBLER|C370|COBOL|
                        LE370|NOT_APPLIC|NOT_DEFINED|PLI),]
      [LOAD_POINT(name4),]
      [LOAD_STATUS(LOADABLE|NOT_APPLIC|NOT_LOADABLE|NOT_LOADED),]
      [LOCATION(CDSA|ECDSA|ELPA|ERDSA|ESDSA|LPA|NONE|RDSA|SDSA),]
      [MODULE_TYPE(MAPSET|PARTITIONSET|PROGRAM),]
      [NEW_PROGRAM_TOKEN(name4),]
      [PROGRAM_ATTRIBUTE(RELOAD|RESIDENT|REUSABLE|TRANSIENT),]
      [PROGRAM_LENGTH(name4),]
      [PROGRAM_TYPE(NOT_APPLIC|PRIVATE|SHARED|TYPE_ANY),]
      [PROGRAM_USAGE(APPLICATION|NUCLEUS),]
      [PROGRAM_USE_COUNT(name4),]
      [PROGRAM_USER_COUNT(name4),]
      [REMOTE_DEFINITION(LOCAL|REMOTE),]
      [REMOTE_PROGID(name8),]
      [REMOTE_SYSID(name4),]
      [REMOTE_TRANID(name4),]
      [SPECIFIED_AMODE(24|31|AMODE_ANY|AMODE_NOT_SPECIFIED|64),]
      [SPECIFIED_RMODE(24|RMODE_ANY|RMODE_NOT_SPECIFIED),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

Note: The options not described in the following list are identical to the equivalent options of the INQUIRE_PROGRAM call. See “The INQUIRE_PROGRAM call” on page 67.

BROWSE_TOKEN(name4)
Specifies a token that identifies the definition to be browsed. This can be either the token returned in the NEW_PROGRAM_TOKEN field of the last GET_NEXT_PROGRAM call, or that in the BROWSE_TOKEN field of the START_BROWSE_PROGRAM call (this token is updated after every GET_PROGRAM call).

name4
The name of a location that contains a 4-byte token.

NEW_PROGRAM_TOKEN(name4)
Returns a token that identifies the next definition in the browse sequence. You can use it in the BROWSE_TOKEN field of your next GET_NEXT_PROGRAM call (or END_BROWSE_PROGRAM call, if you want to end the sequence). You can also use it in the PROGRAM_TOKEN field of INQUIRE_PROGRAM and SET_PROGRAM calls.

name4
The name of a location to receive a 4-byte token that identifies the next program definition.

RESPONSE and REASON values for GET_NEXT_PROGRAM

RESPONSE	***REASON***
OK	None
EXCEPTION	END_LIST
	INVALID_BROWSE_TOKEN
	PROGRAM_NOT_DEFINED_TO_LD
DISASTER	ABEND
	LOCK_ERROR
INVALID	None
KERNERROR	None
PURGED	None

The END_BROWSE_PROGRAM call

END_BROWSE_PROGRAM allows you to end a browse of program definitions initiated by START_BROWSE_PROGRAM.

END_BROWSE_PROGRAM

```
DFHPGISX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(END_BROWSE_PROGRAM),
      BROWSE_TOKEN(name4),]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

BROWSE_TOKEN(name4)
specifies either the token returned in the NEW_PROGRAM_TOKEN field of the last GET_NEXT_PROGRAM call, or that in the BROWSE_TOKEN field of the START_BROWSE_PROGRAM call (this token is updated after every GET_NEXT_PROGRAM call).

RESPONSE and REASON values for END_BROWSE_PROGRAM

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_BROWSE_TOKEN
DISASTER	ABEND
	LOCK_ERROR
INVALID	None
KERNERROR	None
PURGED	None

The INQUIRE_AUTOINSTALL call

INQUIRE_AUTOINSTALL returns information about the current settings of the autoinstall function for programs, mapsets, and partitionsets.

INQUIRE_AUTOINSTALL

```
DFHPGAQX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(INQUIRE_AUTOINSTALL),]
      [OUT,
      [AUTOINSTALL_CATALOG (ALL|MODIFY|NONE),]
      [AUTOINSTALL_EXIT_NAME(name8),]
      [AUTOINSTALL_STATE (ACTIVE|INACTIVE),]
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

AUTOINSTALL_CATALOG(ALL|MODIFY|NONE)
returns the catalog status for autoinstalled program definitions.

ALL
All autoinstalled program, map, and partitionset definitions are cataloged.

MODIFY
Autoinstalled program, map, and partitionset definitions are recorded on the CICS global catalog only if they are modified by a SET PROGRAM command after being autoinstalled.

NONE
No autoinstalled program, map, or partitionset definitions are cataloged.

AUTOINSTALL_EXIT_NAME(name8)
returns the name of the user-replaceable autoinstall control program for programs, mapsets, and partitionsets.

AUTOINSTALL_STATE(ACTIVE|INACTIVE)
returns the status of the program autoinstall function.

ACTIVE
Autoinstall is enabled for programs, mapsets, and partitionsets.

INACTIVE
Autoinstall is not enabled for programs, mapsets, and partitionsets.

RESPONSE and REASON values for INQUIRE_AUTOINSTALL

RESPONSE	*REASON*
OK	None

RESPONSE	*REASON*
EXCEPTION	None
DISASTER	None
INVALID	INVALID_FUNCTION
KERNERROR	None
PURGED	None

The SET_AUTOINSTALL call

SET_AUTOINSTALL enables you to change the settings of the autoinstall function for programs, mapsets, and partitionsets.

SET_AUTOINSTALL

```
DFHPGAQX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(SET_AUTOINSTALL),
      [AUTOINSTALL_CATALOG (ALL|MODIFY|NONE),]
      [AUTOINSTALL_EXIT_NAME(name8),]
      [AUTOINSTALL_STATE (ACTIVE|INACTIVE),]
      [LANGUAGES_AVAILABLE(NO|YES),]]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

AUTOINSTALL_CATALOG(ALL|MODIFY|NONE)
specifies the catalog status for autoinstalled program definitions.

ALL
All autoinstalled program, map, and partitionset definitions are to be cataloged.

MODIFY
Autoinstalled program, map, and partitionset definitions are to be recorded on the CICS global catalog only if they are modified by a SET PROGRAM command after being autoinstalled.

NONE
No autoinstalled program, map, or partitionset definitions are to be cataloged.

AUTOINSTALL_EXIT_NAME(name8)
specifies the name of the user-replaceable autoinstall control program for programs, mapsets, and partitionsets.

AUTOINSTALL_STATE(ACTIVE|INACTIVE)
specifies the status of the program autoinstall function.

ACTIVE
Enable autoinstall for programs, mapsets, and partitionsets.

INACTIVE
Disable autoinstall for programs, mapsets, and partitionsets.

LANGUAGES_AVAILABLE(NO|YES)
specifies whether the autoinstall control program can be called. It can only be called after language establishment.

NO
The control program cannot be called.

YES
The control program can be called.

RESPONSE and REASON values for SET_AUTOINSTALL

RESPONSE	*REASON*
OK	None
EXCEPTION	None
DISASTER	None
INVALID	INVALID_FUNCTION
KERNERROR	None
PURGED	None

The BIND_CHANNEL call

BIND_CHANNEL binds a channel to the task. This call must be issued before the first program in a task.

BIND_CHANNEL

```
DFHPGCHX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(BIND_CHANNEL),
      CHANNEL_TOKEN(name4)]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

CHANNEL_TOKEN(name4)
Specifies the name of a location that contains a 4-byte token representing the channel that is to be bound to the task.

RESPONSE and REASON values for BIND_CHANNEL

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_TOKEN CHANNEL_ALREADY_SET CHANNEL_ON_RESTART
DISASTER	None
INVALID	INVALID_LINK_LEVEL
KERNERROR	None
PURGED	None

Chapter 13. State data access XPI functions

The XPI provides state data access functions that you can use to inquire on, and set, certain system data in the AP domain. These are the DFHAPIQX calls INQ_APPLICATION_DATA, INQUIRE_SYSTEM, and SET_SYSTEM.

The INQ_APPLICATION_DATA call

The INQ_APPLICATION_DATA call enables you to inquire on application system data in the AP domain.

INQ_APPLICATION_DATA

```
DFHAPIQX [CALL,]
   [CLEAR,]
   [IN,
   FUNCTION(INQ_APPLICATION_DATA),]
   [OUT,
   [ACEE(name4 | (Rn) | * ),]     [DSA(name4 | (Rn) | * ),]
   [EIB(name4 | (Rn) | * ),]
   [RSA(name4 | (Rn) | * ),]
   [SYSEIB(name4 | (Rn) | * ),]
   [TCTUA(name4 | (Rn) | * ),]
   [TCTUASIZE(name4 | * ),]
   [TWA(name4 | (Rn) | * ),]
   [TWASIZE(name4 | (Rn) | * ),]
   RESPONSE (name1 | * ),
   REASON (name1 | * )]
```

This command is threadsafe.

ACEE(name4 | (Rn | *)
returns the address of the access control environment element (ACEE).

name4
The name of a fullword area that is to receive the address of the ACEE.

(Rn)
A register that is to receive the ACEE address.

*****
The parameter list itself, in name APIQ_ACEE, is used to hold the address.

DSA(name4 | (Rn | *)
returns the head of the chain of dynamic storage used by application programs to make them reentrant (for example, for assembler programs, the DFHEISTG storage).

name4
The name of a 4-byte area that is to receive the address of the head of the dynamic storage chain.

(Rn)
A register that is to receive the DSA address.

*****
The parameter list itself, in name APIQ_DSA, is used to hold the address.

EIB(name4 | (Rn) | *)
returns the address of the EXEC interface block (EIB) for the current task.

name4
The name of a fullword area that is to receive the address of the EIB.

(Rn)
A register that is to receive the address of the EIB.

*****
The parameter list itself, in name APIQ_EIB, is used to hold the address.

RSA(name4 | (Rn | *)
returns the address of the register save area for the current task.

name4
The name of a fullword area that is to receive the address of the register save area.

(Rn)
A register that is to receive the address of the register save area.

*****
The parameter list itself, name APIQ_RSA, is used to hold the address.

SYSEIB(name4 | (Rn) | *)
returns the address of the system EXEC interface block of the current task.

name4
The name of a fullword area that is to receive the address of the system EXEC interface block.

(Rn)
A register that is to receive the address of the system EXEC interface block.

*****
The parameter list itself, name APIQ_SYSEIB, is used to hold the address.

TCTUA(name4 | (Rn) | *)
returns the address of the terminal control table user area (TCTUA) for the current task.

name4
The name of a fullword area that is to receive the address of the TCTUA.

(Rn)
A register that is to receive the address of the TCTUA.

*****
The parameter list itself, name APIQ_TCTUA, is used to hold the address.

TCTUASIZE(name4 | (Rn) | *)
returns the length in bytes of the TCTUA for the current task.

name4
The name of a 4-byte area that is to receive the length in bytes of the TCTUA.

(Rn)
A register that is to receive the length of the TCTUA.

*****
The parameter list itself, name APIQ_TCTUASIZE, is used to hold the length of the TCTUA.

TWA(name4 | (Rn) | *)
returns the address of the transaction work area.

name4
The name of a fullword area that is to receive the address of the TWA.

(Rn)
A register that is to receive the address of the TWA.

*****
The parameter list itself, name APIQ_TWA, is used to hold the address of the TWA.

TWASIZE(name4 | (Rn) | *)
returns the length, in bytes, of the transaction work area (TWA).

name4
The name of a 4-byte area that is to receive the length, in bytes, of the TWA.

(Rn)
A register that is to receive the length of the TWA.

*****
The parameter list itself, name APIQ_TWASIZE, is used to hold the length of the TWA.

RESPONSE and REASON values for INQ_APPLICATION_DATA

RESPONSE	*REASON*
OK	None
EXCEPTION	DPL_PROGRAM
	NO_TRANSACTION_ENVIRONMENT
	TRANSACTION_DOMAIN_ERROR
DISASTER	ABEND
	LOOP
	INQ_FAILED
INVALID	INVALID_FUNCTION
KERNERROR	None
PURGED	None

The INQUIRE_SYSTEM call

The INQUIRE_SYSTEM call gives you access to CICS system data in the AP domain.

INQUIRE_SYSTEM

```
DFHSAIQX [CALL,]
   [CLEAR,]
   [IN,
   FUNCTION(INQUIRE_SYSTEM),
   [GMMTEXT(name4),]]
   [OUT,
   [CICSREL(name4 | *),]
   [CICSSTATUS(ACTIVE | FINALQUIESCE |
                  FIRSTQUIESCE| INITIALIZING),]
   [CICSSYS(name1 | *),]
   [CICTSLEVEL(name6 | *),]
   [CWA(name4 | (Rn) | *),]
   [CWALENGTH(name2 | *),]
   [DATE(name4|*),]
   [DTRPRGRM(name8 | *),]
   [GMMLENGTH(name2 | *),]
   [GMMTRANID(name4 | *),]
   [INITSTATUS(FIRSTINIT | INITCOMPLETE | SECONDINIT |
                  THIRDINIT),]
   [JOBNAME(name8 | *),]
   [OPREL(name2 | *),]
   [OPSYS(name1 | *),]
   [OSLEVEL(name4 | *),]
   [PLTPI(name2 | *),]
   [SDTRAN(name4 | *),]
   [SECURITYMGR(EXTSECURITY | NOSECURITY),]
   [SHUTSTATUS(CONTROLSHUT | NOTSHUTDOWN | SHUTDOWN),]
   [STARTUP(COLDSTART | EMERGENCY | WARMSTART),]
   [STARTUPDATE(name4| *),]
   [TERMURM(name8 | *),]
   [TIMEOFDAY(name4| *),]
   [XRFSTATUS(NOXRF | PRIMARY | TAKEOVER),]
   RESPONSE (name1 | * ),
   REASON (name1 | * )]
```

This command is threadsafe.

CICSREL(name4 | *)
returns the level number of the CICS code under which the CICS region is running.

name4
The name of a 4-byte location that is to receive the level number characters as hexadecimal values.

CICSSTATUS(ACTIVE|FINALQUIESE|FIRSTQUIESCE|INITIALIZING)
returns the status of the CICS region.

ACTIVE
The CICS region is active and ready to receive work.

FINALQUIESCE
The CICS region is shutting down, and is in the final stage of quiescing.

FIRSTQUIESCE
The CICS region is shutting down, and is in the first stage of quiescing.

INITIALIZING
The CICS region is initializing.

CICSSYS(name1 | *)
returns the operating system for which the running CICS has been built.

name1
The name of a 1-byte area that is to receive the hexadecimal character of the operating system. A value of "X" represents MVS.

CICSTSLEVEL(name6 | *)
returns the release of CICS Transaction Server under which CICS is running.

name6
The name of a 6-byte area that is to receive the release characters as hexadecimal values.

CWA(name4 | (Rn) | *)
returns the address of the common work area.

name4
The name of a 4-byte field that is to receive the address of the CWA.

(Rn)
A register to receive the address of the CWA.

CWALENGTH(name2 | *)
returns the length in bytes of the CWA.

name2
The name of a 2-byte field that is to receive the length of the CWA.

DATE(name4 | *)
returns today's date in packed-decimal form—4-bytes **0Cyyddds**, where:

- **C**is a century indicator. (0=1900, 1=2000, 2=2100, and so on.)
- **yy**=years.
- **ddd**=days.
- **s** is the sign.

name4
The name of a 4-byte location that is to receive the date.

DTRPRGRM(name8 | *)
returns the name of the dynamic routing program.

name8
The name of an 8-byte area that is to receive the name of the dynamic routing program.

GMMLENGTH(name2 | *)
returns the length in bytes of the "good morning" message.

name2
The name of a 2-byte area that is to receive the length of the good morning message.

GMMTEXT(name4)
specifies the address of an area of storage, at least 244 bytes in length and owned by the caller, into which CICS is to return the good morning message.

name4
The address of an area of storage that is to receive the good morning message.

Note: The GMMTEXT parameter must follow the IN statement as an input parameter.

GMMTRANID(name4 | *)
returns the transaction identifier of the CICS good morning transaction.

name4
The name of a 4-byte area that is to receive the CICS good morning transaction id.

INITSTATUS(FIRSTINIT|INITCOMPLETE|SECONDINIT|THIRDINIT)
returns a value indicating the stage reached during CICS initialization.

FIRSTINIT
The first stage of CICS initialization.

INITCOMPLETE
CICS initialization is complete.

SECONDINIT
The second stage of CICS initialization. This stage corresponds to the period when first phase PLTPI programs are run; that is those programs in a PLT that are defined **before** the DFHDELIM statement.

THIRDINIT
The third stage of CICS initialization. This stage corresponds to the period when second phase PLTPI programs are run; that is those programs in a PLT that are defined **after** the DFHDELIM statement.

JOBNAME(name8 | *)
returns the 8-character MVS job name under which the CICS region is running.

name8
The name of a 8-byte area that is to receive the MVS job name.

OPREL(name2 | *)
returns the last 2 digits of the level number of the MVS element of z/OS, under which the CICS region is running.

name2
The name of a 2-byte area that is to receive, as a half-word binary value, the level number of the MVS element of z/OS. For example, z/OS Release 3 MVS is represented by 03.

Note: This field is supported for compatibility purposes only. The information is derived from the last two numbers held in the MVS CVTPRODN field. For example, CVTPRODN holds SP5.2.2 for MVS/ESA SP Version 5 Release 2.2 (in which case OPREL returns 22), and SP6.0.3 for z/OS Release 3. You are recommended to use the OSLEVEL field for the full version and release number of the z/OS product.

OPSYS(name1 | *)
returns the type of operating system on which the CICS regions is running.

name1
The name of a 1-byte area that is to receive the hexadecimal character of the operating system on which CICS is running. A value of "X" represents MVS.

OSLEVEL(name4 | *)
is the version, release, and modification level of the z/OS product on which CICS is running.

name1
The name of a 4-byte area that is to receive the version and release number of z/OS on which CICS is running. A value of "0240" represents z/OS Release 4.

PLTPI(name2 | *)
returns the suffix that identifies the program list table (PLT) containing the list of programs to be run during CICS initialization—the program list table post initialization (PLTPI) list.

name2
The name of a 2-byte area that is to receive the suffix.

SDTRAN(name4 | *)
returns the name of the "shutdown assist" transaction to be run at the beginning of normal or immediate shutdown. The shutdown assist transaction is described in The shutdown assist utility program, DFHCESD.

name4
The name of a 4-byte area to receive the name.

SECURITYMGR(EXTSECURITY|NOSECURITY)
returns a value indicating whether security is active.

EXTSECURITY
CICS is using an external security manager (for example, RACF).

NOSECURITY
Security is not in use in the CICS region—SEC=NO is specified as a system initialization parameter.

SHUTSTATUS(CONTROLSHUT|NOTSHUTDOWN|SHUTDOWN)
returns the shutdown status of the CICS region.

CONTROLSHUT
CICS is performing a controlled shutdown; that is, a normal shutdown with a warm keypoint.

NOTSHUTDOWN
CICS is not in shutdown mode.

SHUTDOWN
CICS is performing an immediate shutdown.

STARTUP(COLDSTART|EMERGENCY|WARMSTART)
returns the type of startup the CICS region performed.

COLDSTART
CICS performed a cold start, either because this was explicitly specified on the system initialization parameter, or because CICS forced a cold start because of the state of the global catalog.

EMERGENCY
CICS performed an emergency restart because the previous run did not shut down normally with a warm keypoint.

WARMSTART
CICS performed a warm restart following the normal shutdown of the previous run.

STARTUPDATE(name4 | *)
returns the start-up-date of this CICS region, in packed decimal form (4-bytes **00yydddc** where **yy**=years, **ddd**=days, **c** is the sign).

name4
The name of a 4-byte location that is to receive the startup date of this CICS system.

TERMURM(name8 | *)
returns the name of the autoinstall user program for terminals.

name8
The name of an 8-byte area that is to receive the name of the autoinstall user program for terminals.

TIMEOFDAY(name4 | *)
returns the current time-of-day in packed decimal form (4-bytes **hhmmsstc** where **hh**=hours, **mm**=minutes, **ss**=seconds, **t**=tenths of a second, and **c** is the sign).

name4
The name of a 4-byte location that is to receive the time.

XRFSTATUS(NOXRF|PRIMARY|TAKEOVER)
returns the XRF status of the CICS region.

NOXRF
CICS was started with the system initialization parameter XRF=NO specified. XRF is not active.

PRIMARY
The CICS region was started as an active CICS in an XRF environment.

TAKEOVER
The CICS region was started as an alternate CICS, with the START=STANDBY system initialization parameter.

RESPONSE and REASON values for INQUIRE_SYSTEM

RESPONSE	*REASON*
OK	None
INVALID	INVALID_FUNCTION
EXCEPTION	LENGTH_ERROR
	UNKNOWN_DATA
DISASTER	INQ_FAILED
PURGED	None

The SET_SYSTEM call

The SET_SYSTEM call allows you to set CICS system data values in the AP domain.

SET_SYSTEM

```
DFHSAIQX [CALL,]
   [CLEAR,]
   [IN,
   FUNCTION(SET_SYSTEM),
   [DTRPRGRM(name8 | string | 'string'),]
   [GMMLENGTH(name2 | (Rn) | expression),]
   [GMMTEXT(name8 | (Rn)),]]
   [OUT,
   RESPONSE (name1 | * ),
   REASON (name1 | * )]
```

This command is threadsafe.

DTRPRGRM(name8 | string | 'string')
specifies the name of the dynamic routing program.

name8
The name of an 8-byte area that contains the name of the dynamic routing program.

string
A string of character, without intervening blanks, that defines the name of the dynamic routing program being set.

‘string’
A string of character without intervening blanks. If you want to document a name (label) in your program, use this form.

GMMLENGTH(name2 | (Rn))
specifies the length of the new "good morning" message supplied by the GMMTEXT parameter.

name2
The name of a 2-byte area that contains, as a half-word binary value, the length of the new good morning message.

(Rn)
A register that contains the length of the new good morning message.

GMMTEXT(name4 | (Rn))
specifies the new good morning message.

name4
The name of a 4-byte location that contains the address of a storage area (up to a maximum of 246 bytes long) that contains the good morning message.

(Rn)
A register that contains the address of a storage area (up to a maximum of 246 bytes long) that contains the good morning message.

RESPONSE and REASON values for SET_SYSTEM

RESPONSE	***REASON***
OK	None
INVALID	INVALID_FUNCTION
EXCEPTION	AKP_SIZE_ERROR
	NO_KEYPOINT
DISASTER	SET_FAILED
PURGED	None

Chapter 14. Storage control XPI functions

The XPI provides seven storage control functions. These are the DFHSMMCX macro calls GETMAIN, FREEMAIN, INQUIRE_ELEMENT_LENGTH, and INQUIRE_TASK_STORAGE, and the DFHSMSRX calls INQUIRE_ACCESS, INQUIRE_SHORT_ON_STORAGE, and SWITCH_SUBSPACE.

DFHSMMCX calls cannot be used in any exit program invoked from any global user exit point in the following domains or program:

- Dispatcher domain
- Dump domain
- Monitor domain
- Statistics domain
- Transient data program.

The GETMAIN call

GETMAIN acquires an element of storage for use by your exit program. You can ask for a specific CLASS of storage, and you can request that it is initialized to a single-byte value.

Storage that is acquired by using a GETMAIN call and that is in the following classes is released by CICS when the TCA being used at the time of the acquisition terminates:

- CICS
- CICS24
- USER
- USER24.

In contrast, storage in the following classes is not released automatically at task-end. You must use the FREEMAIN call to release it.

- SHARED_CICS
- SHARED_CICS24
- SHARED_USER
- SHARED_USER24
- TERMINAL.

Also, some user exits can be invoked from system tasks. In these circumstances, storage is not released until the next CICS shutdown. Therefore, use the FREEMAIN call to release all storage areas acquired by a GETMAIN call as soon as you finish using them.

GETMAIN

```
DFHSMMCX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(GETMAIN),
      GET_LENGTH(name4 | (Rn) | expression),
      STORAGE_CLASS(CICS|CICS24|SHARED_CICS|SHARED_CICS24|
                  SHARED_USER|SHARED_USER24|USER|USER24|TERMINAL),
      SUSPEND(NO|YES),
      [INITIAL_IMAGE(name1 | literalconst),]
      [TCTTE_ADDRESS(name4 | (Ra)),]]
      [OUT,
      ADDRESS(name4 | (Rn) | *),
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

ADDRESS(name4 | (Rn) | *)
Returns the address of the storage obtained by the call.

name4
The name of a fullword where the obtained storage address is saved.

(Rn)
A register that is set to point to the obtained storage.

*****
The parameter list itself, name SMMC_ADDRESS, is used to keep the address.

GET_LENGTH(name4 | (Rn) | expression)
Specifies the number of bytes of storage you want, expressed in any of the following ways:

name4
The name of a fullword specifying, in binary, the number of bytes.

(Rn)
A register containing, in binary, the number of bytes.

expression
A valid assembler-language expression; for example, a number, a symbolic expression, or a combination of the two.

If you request TERMINAL storage, the length you specify does not include the length of the storage accounting area (SAA). The maximum length you can specify is 65,515 bytes. CICS storage management adds an 8-byte SAA, and the address returned by the XPI call is that of the start of the SAA.

If you request CICS24, CICS, USER24, USER, SHARED_CICS24, SHARED_CICS, SHARED_USER24, or SHARED_USER storage, you need only specify the length needed by your program. The address returned is that of the start of your data storage. The maximum size of storage for these storage classes is the same as the size of the DSA from which they are allocated.

INITIAL_IMAGE(name1 | literalconst)
Specifies the initializing pattern. For example, you might want to set the acquired storage to binary zeros.

name1
The name of a location where the one-byte initializing pattern is stored.

literalconst
A number in the form of a literal, for example B'00000000', X'FF', X'FC', "0", or an equate symbol with a similar value.

STORAGE_CLASS(CICS|CICS24|SHARED_CICS|SHARED_CICS24| SHARED_USER|SHARED_USER24| USER|USER24|TERMINAL)
Specifies the class of the storage that is the subject of the call. The values you can assign to this option, and the type of storage each represents, are listed in Table 6 on page 94.

Table 6. CICS storage classes

STORAGE_CLASS	Type of storage
CICS	Task-lifetime CICS-key storage above 16 MB but below 2 GB
CICS24	Task-lifetime CICS-key storage below 16 MB
SHARED_CICS	Shared CICS-key storage above 16 MB but below 2 GB
SHARED_CICS24	Shared CICS-key storage below 16 MB
SHARED_USER	Shared user-key storage above 16 MB but below 2 GB
SHARED_USER24	Shared user-key storage below 16 MB
TERMINAL	This class of storage has an 8-byte SAA.

Table 6. CICS storage classes (continued)	
STORAGE_CLASS	**Type of storage**
USER	Task-lifetime user-key storage above 16 MB but below 2 GB
USER24	Task-lifetime user-key storage below 16 MB

You must specify a storage class on a GETMAIN request. On a FREEMAIN request it is an optional parameter, and any value that you specify is not checked by CICS.

SUSPEND(YES|NO)
Specifies whether to suspend your request if there is less storage available than you requested in the GET_LENGTH option.

TCTTE_ADDRESS(name4 | (Ra))
Specifies the address of the terminal control table terminal entry (TCTTE). On GETMAIN requests, you must code this option if you specify a class of TERMINAL on the STORAGE_CLASS option. On FREEMAIN requests, you must code this option if you release TERMINAL-class storage.

Note: Before you obtain TERMINAL class storage, check TCAFCI bit 7 to ensure that the TCA is running under a terminal.

name4
The name of a fullword containing the address.

(Ra)
A register that points to the TCTTE.

RESPONSE and REASON values for GETMAIN

RESPONSE	***REASON***
OK	None
EXCEPTION	INSUFFICIENT_STORAGE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note:

1. For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.
2. INSUFFICIENT_STORAGE is returned if the GETMAIN request was specified with SUSPEND(NO), and there was not enough storage available to satisfy the request.
3. PURGED is returned if the GETMAIN request was specified with SUSPEND(YES), there was not enough storage to satisfy the request, and the task was purged.

The FREEMAIN call

FREEMAIN releases an area of storage that is currently allocated to your exit program.

FREEMAIN

```
DFHSMMCX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(FREEMAIN),
      ADDRESS(name4 | (Rn) | *),
```

```
      [STORAGE_CLASS(CICS|CICS24|SHARED_CICS|SHARED_CICS24|
                  SHARED_USER|SHARED_USER24|USER|USER24|TERMINAL),]
      [TCTTE_ADDRESS(pointer),]]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

For an explanation of the options, see "The GETMAIN call" on page 93.

RESPONSE and REASON values for FREEMAIN

RESPONSE	*REASON*
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

The INQUIRE_ACCESS call

INQUIRE_ACCESS returns the access-key of an element of storage specified by start address and length. If the element is not wholly contained within one of the CICS dynamic storage areas (DSAs), CICS returns an exception response.

INQUIRE_ACCESS

```
DFHSMSRX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(INQUIRE_ACCESS),
      ELEMENT_ADDRESS(name4 | (Rn) | *),
      ELEMENT_LENGTH(name4 | (Rn) | *),]
      [OUT,
      ACCESS(CICS | READ_ONLY | USER),
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

ACCESS(CICS|READ_ONLY|USER)
returns the access-key of the storage element.

CICS
CICS-key

READ_ONLY
Readonly storage

USER
User-key.

ELEMENT_ADDRESS(name4 | (Rn) | *)
specifies the address of the storage element.

ELEMENT_LENGTH(name4 | (Rn) | *)
specifies the length of the storage element, in bytes. A length of zero is treated as a length of one.

RESPONSE and REASON values for INQUIRE_ACCESS

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_ELEMENT
DISASTER	None
INVALID	None
KERNERROR	None

The INQUIRE_ELEMENT_LENGTH call

INQUIRE_ELEMENT_LENGTH enables you to pass the address of any part of an element of task-lifetime storage, and to obtain from CICS the start address and the length of the storage element that contains the passed address.

INQUIRE_ELEMENT_LENGTH

```
DFHSMMCX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION (INQUIRE_ELEMENT_LENGTH),
       ADDRESS (name4 | (Rn) | *),]
       [OUT,
       ELEMENT_ADDRESS(name4 | (Rn) | *),
       ELEMENT_LENGTH(name4 | (Rn) | *),
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

ADDRESS(name4 | (Rn) | *)
specifies an address that lies within an element of task-lifetime storage of the current task.

CICS accepts addresses that reference the leading or trailing check zones as being valid addresses for the element of storage you are inquiring upon.

ELEMENT_ADDRESS(name4 | (Rn) | *)
returns the start address of the element of task-lifetime storage referenced by the ADDRESS parameter. The start address returned does **not** include the leading check zone.

ELEMENT_LENGTH(name4 | (Rn) | *)
returns the length of the element of task-lifetime storage referenced by the ADDRESS parameter. The length returned does **not** include the leading or trailing check zones.

RESPONSE and REASON values for INQUIRE_ELEMENT_LENGTH

RESPONSE	*REASON*
OK	None
EXCEPTION	INVALID_ADDRESS
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The INQUIRE_SHORT_ON_STORAGE call

INQUIRE_SHORT_ON_STORAGE enables you to determine whether CICS is short on 64-bit (above-the-bar) storage, short on storage above 16 MB but below 2 GB (above the line), or short on storage below 16 MB (below the line).

INQUIRE_SHORT_ON_STORAGE

```
DFHSMSRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(INQUIRE_SHORT_ON_STORAGE),]
       [OUT,
       SOS_ABOVE_THE_BAR(NO|YES),
       SOS_ABOVE_THE_LINE(NO|YES),
       SOS_BELOW_THE_LINE(NO|YES),
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

SOS_ABOVE_THE_BAR(NO|YES),
Returns YES if CICS is currently short on 64-bit (above-the-bar) storage, and NO if not.

SOS_ABOVE_THE_LINE(NO|YES),
Returns YES if CICS is currently short on storage above 16 MB but below 2 GB, and NO if not.

SOS_BELOW_THE_LINE(NO|YES),
returns YES if CICS is currently short on storage below 16 MB, and NO if not.

RESPONSE and REASON values for INQUIRE_SHORT_ON_STORAGE

RESPONSE	*REASON*
OK	None
DISASTER	None
KERNERROR	None

The INQUIRE_TASK_STORAGE call

INQUIRE_TASK_STORAGE enables you to request details of all elements of task-lifetime storage belonging to a task. You can specify the transaction number of the task explicitly on the call, or let it default to the current task.

INQUIRE_TASK_STORAGE

```
DFHSMMCX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION (INQUIRE_TASK_STORAGE),
       [TRANSACTION_NUMBER(name4 | (Rn) | *),]
       ELEMENT_BUFFER(buffer-descriptor),
       LENGTH_BUFFER(buffer-descriptor),]
       [OUT,
       NUMBER_OF_ELEMENTS(name4 | (Rn) | *),
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

ELEMENT_BUFFER(buffer-descriptor)
defines the address and length of a buffer into which CICS returns a list of start addresses of all the elements of task-lifetime storage belonging to either the specified task or, by default, the current task.

The start addresses returned do **not** include the leading check zone. For a description of a buffer descriptor, see XPI syntax.

LENGTH_BUFFER(buffer-descriptor)
defines the address and length of a buffer into which CICS returns a list of the lengths of the elements of task-lifetime storage belonging to either the specified task or, by default, the current task. The lengths returned do **not** include the leading or trailing check zones.

For a description of a buffer descriptor, see XPI syntax.

NUMBER_OF_ELEMENTS(name4 | (Rn) | *)
returns the number of entries in each of the two buffers, ELEMENT_BUFFER and LENGTH_BUFFER, as a full-word binary value.

TRANSACTION_NUMBER(name4 | (Rn) | *)
specifies, as a 4 byte packed decimal value, the transaction number of the task to whom the storage belongs.

If you omit the transaction (task) number, CICS assumes the current task.

RESPONSE and REASON values for INQUIRE_TASK_STORAGE

RESPONSE	*REASON*
OK	None
EXCEPTION	INSUFFICIENT_STORAGE
	NO_TRANSACTION_ENVIRONMENT
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The SWITCH_SUBSPACE call

SWITCH_SUBSPACE causes CICS to switch from a subspace to base space, if the task is not already executing in the base space. If the task is already in the base space, storage manager ignores the call.

This function can be used by global user exit programs that receive control in subspace and for some reason need to switch into basespace.

SWITCH_SUBSPACE

```
DFHSMSRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(SWITCH_SUBSPACE),
       SPACE(BASESPACE),]
       [OUT,
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

SPACE(BASESPACE)
specifies that CICS is to switch the task issuing the call to the basespace, if it is currently executing within a subspace. This enables the task to read and write to another task's user-key task-lifetime storage.

RESPONSE and REASON values for SWITCH_SUBSPACE

RESPONSE	*REASON*
OK	None
DISASTER	None
KERNERROR	None

Chapter 15. Trace control XPI function

The XPI provides one trace control function. This is the DFHTRPTX call TRACE_PUT.

Restriction: DFHTRPTX calls cannot be used in any exit program invoked from any global user exit point in the:

- Dispatcher domain
- Dump domain
- Monitor domain
- Statistics domain
- Transient data program.

The TRACE_PUT call

TRACE_PUT writes a trace entry to the active trace destinations.

Only make a TRACE_PUT call when UEPTRON indicates that tracing is active for the function containing the exit program (see UEPTRON in DFHUEPAR). You might prefer to make exception trace entries, in case of serious errors, without testing UEPTRON.

If you use TRACE_PUT to write exception trace entries, identify these so they are highlighted as exception trace entries by the trace formatting utility program. To identify an exception trace entry, enter the literal string 'USEREXC' in the DATA1 block descriptor field on the DFHTRPTX call.

TRACE_PUT

```
DFHTRPTX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(TRACE_PUT),
      POINT_ID(literalconst | name2 | (Rn)),
      [DATA1(block-descriptor),]
      [DATA2(block-descriptor),]
      [DATA3(block-descriptor),]
      [DATA4(block-descriptor),]
      [DATA5(block-descriptor),]
      [DATA6(block-descriptor),]
      [DATA7(block-descriptor),]
      [RETURN_ADDR(expression | name4 | (Ra)),]]
      [OUT,
      RESPONSE(name1 | *)]
```

This command is threadsafe.

DATA*n*(block-descriptor)
Specifies up to seven areas to be included in the data section of the trace entry. For a description of valid block-descriptors, see XPI syntax. If you specify any given DATA*n*, then DATA1 through DATA(*n*-1) must be coded before DATA*n*. The specified DATA items are printed in the trace output in the order specified, that is, in order of DATA1 through DATA*n*. A 2-byte length field is printed before the data field itself. The maximum total length of the data that can be traced in one call is 4000 bytes. The total length of all the data fields and all their 2-byte length fields must be within this limit.

POINT_ID(literalconst|name2|(Rn))
Specifies the trace entries made as a result of this request. Every TRACE_PUT call within a calling domain should specify a unique POINT_ID. This enables you to locate the origin of a trace call when examining a formatted trace. The POINT_IDs must be in the range decimal 256 through 511 (X'100' through X'1FF'). This range is not used in CICS modules, but is reserved for user exits.

literalconst
A number in the form of a literal, containing the ID

name2
The name of a 2-byte field containing the ID

(Rn)
A register, the two low-order bytes of which contain the ID.

RETURN_ADDR(expression|name4|(Ra))
Specifies the value that appears in the return address field of the trace entry.

expression
A valid assembler-language expression that results in the address

name4
The name of a fullword containing the address

(Ra)
A register containing the address.

Chapter 16. Transaction management XPI functions

The XPI provides transaction management functions that you can use to inquire about transactions and set certain transaction parameters.

The INQUIRE_CONTEXT call

INQUIRE_CONTEXT returns information about the environment in which a transaction is running. Specifically, it provides information for transactions running in a bridge environment.

INQUIRE_CONTEXT

```
DFHBRIQX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(INQUIRE_CONTEXT),]
       [OUT,
       [CONTEXT(byte1),]
       [BRIDGE_TRANSACTION_ID(name4),]
       [BRIDGE_EXIT_PROGRAM(name8),]
       [BFB_TOKEN(name4),]
       [BRXA_TOKEN(name4),]
       [FACILITYTOKEN(name8),]
       [START_TYPE(byte1),]
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

BFB_TOKEN(name4)
returns a pointer that contains the address of the bridge facility used by this task. Although the bridge facility is not a real terminal, it is represented by a data structure that has the same format as a TCTTE and can be mapped using the DSECT DFHTCTTE. If CONTEXT returns NORMAL, the contents of this field are meaningless.

Note: In earlier releases of CICS, this field was called BRIDGE_FACILITY_TOKEN.

name4
The name of a 4-byte location to receive the token.

BRIDGE_EXIT_PROGRAM(name8)
returns the name of the bridge exit program used by this task. If CONTEXT returns NORMAL, the contents of this field are meaningless.

name8
The name of an 8-byte location to receive the name of the bridge exit program.

BRIDGE_TRANSACTION_ID(name4)
returns the name of the bridge monitor transaction that issued a START BREXIT TRANSID command to start this transaction. If CONTEXT returns NORMAL, the contents of this field are meaningless.

name4
The name of a 4-byte location to receive the name of the bridge monitor transaction.

BRXA_TOKEN(name4)
returns a token that contains the address of the bridge exit area (BRXA) used by this task. The BRXA is not applicable to the Link3270 bridge (START_TYPE=BRIQ_LINK). The format of BRXA is defined by the DFHBRARx copy books. If CONTEXT returns NORMAL, the contents of this field are meaningless.

name4
The name of a 4-byte location to receive the token.

CONTEXT(byte1)
returns, in a 1-byte location (*byte1*), the type of environment in which the transaction is running.

BRIDGE
A user transaction that was started using a bridge

BREXIT
A bridge exit program

NORMAL
A transaction that is not running in a bridge environment.

FACILITYTOKEN(name8)
returns the facilitytoken (an identifier associated with the bridge facility). If CONTEXT returns NORMAL, the contents of this field are meaningless.

name8
The name of an 8-byte location to receive the facilitytoken.

START_TYPE(byte1)
returns, in a 1–byte location (*byte1*), an indication of how the 3270 bridge was started. If CONTEXT returns NORMAL, the contents of this field are meaningless.

BRIQ_START
The bridge was started using START BREXIT.

BRIQ_LINK
The bridge was started using the Link3270 mechanism.

RESPONSE and REASON values for INQUIRE_CONTEXT

RESPONSE	*REASON*
OK	None
DISASTER	ABEND
	LOOP
INVALID	None
EXCEPTION	NO_TRANSACTION_ENVIRONMENT
KERNERROR	None

The INQUIRE_DTRTRAN call

INQUIRE_DTRTRAN returns the name of the dynamic transaction routing (DTR) transaction definition.

The DTR transaction definition provides common attributes for transactions that are to be dynamically routed and which do not have a specific transaction definition. It is specified on the DTRTRAN system initialization parameter; the CICS-supplied default definition is CRTX.

INQUIRE_DTRTRAN

```
DFHXMSRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(INQUIRE_DTRTRAN),]
       [OUT,
       DTRTRAN(name4),
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

DTRTRAN(name4)
returns the name of the DTR transaction definition to used for routing transactions that are not defined by an explicit transaction resource definition.

name4
The name of a 4-byte location that is to receive the name of the DTR transaction definition. If 'NO' was specified on the DTRTRAN system initialization parameter, 'NO' will be placed in this field.

RESPONSE and REASON values for INQUIRE_DTRTRAN

RESPONSE	*REASON*
OK	None
DISASTER	ABEND
	LOGIC_ERROR
	LOOP
INVALID	INVALID_FUNCTION
KERNERROR	None
PURGED	None

The INQUIRE_MXT call

The INQUIRE_MXT function is provided on the DFHXMSRX macro call. Its purpose is to provide current value of the MXT parameter.

INQUIRE_MXT

```
DFHXMSRX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(INQUIRE_MXT),]
       [OUT,
       CURRENT_ACTIVE(name4 | (Rn) ),
       MXT_LIMIT(name4 | (Rn)),
       MXT_QUEUED(name4 | (Rn) ),
       TCLASS_QUEUED(name4 | (Rn) ),
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

CURRENT_ACTIVE(name4 | (Rn))
returns the current number of all active user tasks.

name4
The name of a 4-byte location that is to receive the current number of active user tasks, expressed as a binary value.

(Rn)
A register to receive the current number of active user tasks, expressed as a binary value.

MXT_LIMIT(name4 | (Rn))
returns the current number of the MXT parameter.

name4
The name of a 4-byte location that is to receive the maximum number of all user tasks currently allowed, expressed as a binary value.

(Rn)
A register to receive the maximum number of all tasks currently allowed, expressed as a binary value.

MXT_QUEUED(name4 | (Rn))
returns the current number of user transactions that are queued as a result of the maximum tasks (MXT) being reached.

name4
The name of a 4-byte location that is to receive the current number of queued user tasks, expressed as a binary value.

(Rn)
A register to receive the current number of queued user tasks, expressed as a binary value.

TCLASS_QUEUED(name4 | (Rn))
returns the current number of all transactions that are queued for transaction class membership.

name4
The name of a 4-byte location that is to receive the current number of queued transaction class members, expressed as a binary value.

(Rn)
A register to receive the current number of queued transaction class members, expressed as a binary value.

RESPONSE and REASON values for INQUIRE_MXT

RESPONSE	*REASON*
OK	None
DISASTER	LOGIC_ERROR
	ABEND
	LOOP
INVALID	INVALID_FUNCTION
KERNERROR	None
PURGED	None

The INQUIRE_TCLASS call

The INQUIRE_TCLASS function is provided on the DFHXMCLX macro call. Its purpose is to provide current information about the specified transaction class (TCLASS).

INQUIRE_TCLASS

```
DFHXMCLX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(INQUIRE_TCLASS),
       INQ_TCLASS_NAME(name8 | string | 'string'),]
       [OUT,
       [CURRENT_ACTIVE(name4 | (Rn)),]
       [CURRENT_QUEUED(name4 | (Rn)),]
       [MAX_ACTIVE(name4 | (Rn)),]
       [PURGE_THRESHOLD(name4 | (Rn)),]
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

CURRENT_ACTIVE(name4 | (Rn))
returns the current number of active user tasks in this transaction class.

name4
The name of a 4-byte location that is to receive the current number of active user tasks for this transaction class, expressed as a binary value.

(Rn)
A register to receive the current number of active user tasks for this transaction class, expressed as a binary value.

CURRENT_QUEUED(name4 | (Rn))
returns the current number of queued user tasks.

name4
The name of a 4-byte location that is to receive the current number of queued user tasks in this transaction class, expressed as a binary value.

(Rn)
A register to receive the current number of queued user tasks, expressed as a binary value.

INQ_TCLASS_NAME(name8 | string | 'string')
specifies the name of the transaction class for this inquiry.

name8
The name of an 8-byte location that contains the name of the transaction class being inquired on.

string
A string of characters, without intervening blanks, naming the transaction class.

'string'
A string of characters, within quotation marks, naming the transaction class. The string length is set to 8 by padding with blanks within the quotation marks.

MAX_ACTIVE(name4 | (Rn))
returns the current maximum number of active tasks allowed for the transaction class.

name4
The name of a 4-byte location that is to receive the current maximum number of active tasks currently allowed for this transaction class, expressed as a binary value.

(Rn)
A register to receive the current maximum number of active tasks currently allowed for this transaction class, expressed as a binary value.

PURGE_THRESHOLD(name4 | (Rn))
returns the purge threshold limit for this transaction class.

name4
The name of a 4-byte location that is to receive the current purge threshold limit for this transaction class, expressed as a binary value.

(Rn)
A register to receive the current purge threshold limit for this transaction class, expressed as a binary value.

RESPONSE and REASON values for INQUIRE_TCLASS

RESPONSE	*REASON*
OK	None
DISASTER	LOGIC_ERROR
INVALID	None
EXCEPTION	UNKNOWN_CLASS

The INQUIRE_TRANDEF call

The INQUIRE_TRANDEF function is provided on the DFHXMXDX macro call. Its purpose is to allow you to obtain information about the specified transaction definition. In general, this function call is equivalent to the EXEC CICS INQUIRE TRANSACTION command, with some differences.

INQUIRE_TRANDEF

```
DFHXMXDX  [CALL,]
       [CLEAR,]
       [IN,
```

```
        FUNCTION(INQUIRE_TRANDEF),
        INQ_TRANSACTION_ID(name4 | string | 'string'),]
        [OUT,
        [BREXIT(name8),]
        [CMDSEC(name1),]
        [DTIMEOUT(name4 | (Rn)),]
        [DUMP(name1),]
        [DYNAMIC(name1),]
        [INDOUBT(name1),]
        [INDOUBT_WAIT(name1),]
        [INDOUBT_WAIT_TIME(name4),]
        [INITIAL_PROGRAM(name8),]
        [ISOLATE(name1),]
        [LOCAL_QUEUING(name1),]
        [OTSTIMEOUT(name4 | (Rn)),]
        [PARTITIONSET(name1),]
        [PARTITIONSET_NAME(name8),]
        [PROFILE_NAME(name8),]
        [REMOTE(name1),]
        [REMOTE_NAME(name8),]
        [REMOTE_SYSTEM(name4),]
        [RESSEC(name1),]
        [RESTART(name1),]
        [ROUTABLE_STATUS(ROUTABLE|NOT_ROUTABLE),]
        [RUNAWAY_LIMIT(name4 | (Rn)),]
        [SHUTDOWN(name1),]
        [SPURGE(name1),]
        [STATUS(name1),]
        [STORAGE_CLEAR(name1),]
        [STORAGE_FREEZE(name1),]
        [SYSTEM_ATTACH(name1),]
        [SYSTEM_RUNAWAY(name1),]
        [TASKDATAKEY(name1),]
        [TASKDATALOC(name1),]
        [TCLASS(name1),[TCLASS_NAME(name8),]]
        [TPURGE(name1),]
        [TRACE(name1),]
        [TRAN_PRIORITY(name4 | (Rn)),]
        [TRAN_ROUTING_PROFILE(name8),]
        [TRANSACTION_ID(name4),]
        [TWASIZE(name4 | (Rn)),]
        RESPONSE (name1 | *),
        REASON (name1 | *)]
```

This command is threadsafe.

The following parameter descriptions explain briefly the possible values that can be returned on an INQUIRE_TRANDEF call. For a more detailed explanation of some of these parameters, see the corresponding parameter descriptions for the TRANSACTION resource definition in TRANSACTION attributes.

BREXIT(name8)
returns the name of the default bridge exit program specified for the named transaction. If no bridge exit is specified, blanks are returned.

name8
The name of an 8-byte location to receive the name of the bridge exit program.

CMDSEC(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether command security checking is required for the transaction.

XMXD_YES
Command security checking is required.

XMXD_NO
Command security checking is not required.

DTIMEOUT(name4)
returns the deadlock timeout value for the transaction.

name4
The name of a 4-byte location that is to receive the deadlock timeout value, expressed as a binary value.

(Rn)
A register to receive the deadlock timeout value, expressed as a binary value.

Note that a value of zero means that the transaction resource definition specifies DTIMOUT(NO).

DUMP(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether CICS is to take a transaction dump if the transaction abends.

XMXD_YES
A transaction dump is required.

XMXD_NO
A transaction dump is not required.

DYNAMIC(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction is defined for dynamic transaction routing.

XMXD_YES
The transaction is to be dynamically routed to a remote CICS.

XMXD_NO
The transaction is not to be dynamically routed.

INDOUBT(name1)
returns, in a 1-byte location (*name1*), an equated value indicating the action to be taken if the CICS region fails or loses connectivity with its coordinator while a unit of work is in the indoubt period. (The action is based on the ACTION attribute of the TRANSACTION resource definition.)

The action is dependent on the values returned in INDOUBT_WAIT and INDOUBT_WAIT_TIME; if INDOUBT_WAIT returns XMXD_YES, the action is not taken until the time returned in INDOUBT_WAIT_TIME expires.

XMXD_BACKOUT
Any changes made by the transaction to recoverable resources are to be backed out.

XMXD_COMMIT
Any changes made by the transaction to recoverable resources are to be committed.

INDOUBT_WAIT(name1)
returns, in a 1-byte location (*name1*), an equated value indicating how a unit of work (UOW) is to respond if a failure occurs while it is in an indoubt state.

XMXD_NO
The UOW is not to wait, pending recovery from the failure. CICS is to take immediately whatever action is specified on the ACTION attribute of the TRANSACTION definition.

XMXD_YES
The UOW is to wait, pending recovery from the failure, to determine whether recoverable resources are to be backed out or committed.

INDOUBT_WAIT_TIME(name4)
returns the length of time, in minutes, after a failure during the indoubt period, before the transaction is to take the action returned in the INDOUBT field. The returned value is valid only if the unit of work is indoubt and INDOUBT_WAIT returns XMXD_YES.

name4
The name of a 4-byte location that is to receive the delay time, expressed as a binary value.

See also INDOUBT and INDOUBT_WAIT.

INITIAL_PROGRAM(name8)
returns the name of the initial program to be given control for the transaction.

name8
The name of an 8-byte location to receive the initial program name.

INQ_TRANSACTION_ID(name4 | string | ‘string')
specifies the transaction identifier for this transaction definition inquiry.

name4
The name of a 4-byte location that contains the name of the transaction identifier.

string
A string of characters, without intervening blanks, naming the transaction identifier.

‘string'
A string of characters, within quotation marks, naming the transaction identifier. The string length is set to 4 by padding with blanks within the quotation marks.

ISOLATE(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether transaction isolation is required for the transaction's task-lifetime user-key storage.

XMXD_NO
Transaction isolation is not required for task-lifetime user-key storage.

XMXD_YES
Transaction isolation is required for task-lifetime user-key storage.

LOCAL_QUEUING(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether a start request for this transaction is eligible to queue locally if the transaction is to be started on another system, and the remote system is not available.

XMXD_NO
The request is not to be queued locally.

XMXD_YES
The request can be queued locally.

OTSTIMEOUT(name4)
returns the default period in seconds that an Object Transaction Service (OTS) transaction created in an Enterprise JavaBeans (EJB) environment and executing under this CICS transaction is allowed to execute without the initiator of the OTS transaction taking a syncpoint (or rolling back the OTS transaction).

name4
The name of a 4-byte location to receive the timeout setting, expressed as a binary value.

(Rn)
A register to receive the timeout setting, expressed as a binary value.

A value of zero means that the transaction resource definition specifies OTSTIMEOUT(NO).

PARTITIONSET(name1)
returns, in a 1-byte location (*name1*), an equated value indicating the partitionset specified on the transaction definition.

XMXD_KEEP
The reserved name KEEP is specified for the partitionset, which means tasks running under this transaction definition use the application partitionset for the terminal associated with the transaction.

XMXD_NAMED
The partitionset is named specifically on the transaction definition. The name is returned on the PARTITIONSET_NAME parameter.

XMXD_NONE
There is no partitionset specified for the transaction definition.

XMXD_OWN
The reserved name OWN is specified for the partitionset, which means tasks running under this transaction definition perform their own partitionset management.

PARTITIONSET_NAME(name8)
returns the name of the partitionset defined on the transaction definition.

name8
The name of an 8-byte location that is to receive the name of the partitionset.

PROFILE_NAME(name8)
returns the name of the profile definition that is associated with the transaction definition.

name8
The name of an 8-byte location to receive the name of the profile definition associated with the transaction definition.

REMOTE(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction is defined as remote.

XMXD_NO
The transaction is not a remote transaction.

XMXD_YES
The transaction is a remote transaction.

REMOTE_NAME(name8)
returns the name by which the transaction is known in a remote system.

name8
The name of an 8-byte location to receive the transaction's remote name.

REMOTE_SYSTEM(name4)
returns the name of the remote system as specified on the transaction definition.

If the DYNAMIC parameter returns XMXD_YES, REMOTE_SYSTEM returns the default name, which can be changed by the dynamic routing program.

If the DYNAMIC parameter returns XMXD_NO, this is the actual remote system to which the transaction is to be routed.

name4
The name of a 4-byte location to receive the defined name of the remote system.

RESSEC(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether resource security checking is required for the transaction.

XMXD_NO
Resource security checking is not required.

XMXD_YES
Resource security checking is required.

RESTART(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction is to be considered for transaction restart.

XMXD_NO
The transaction cannot be restarted.

XMXD_YES
The transaction can be restarted.

ROUTABLE_STATUS(ROUTABLE|NOT_ROUTABLE)
returns a value indicating whether, if the transaction is the subject of an eligible EXEC CICS START command, it will be routed using the enhanced routing method.

NOT_ROUTABLE
If the transaction is the subject of a START command, it will be routed using the “traditional” method.

ROUTABLE
If the transaction is the subject of an eligible START command, it will be routed using the enhanced method.

For details of the enhanced and "traditional" methods of routing transactions invoked by EXEC CICS START commands, see CICS transaction routing.

RUNAWAY_LIMIT(name4 | (Rn))
returns the runaway-task time limit specified on the transaction definition. If SYSTEM_RUNAWAY is XMXD_YES, the value returned is the value defined by the **ICVR** system initialization parameter.

name4
The name of a 4-byte location that is to receive the task runaway limit, expressed as a binary value.

(Rn)
A register to receive the task runaway limit, expressed as a binary value.

SHUTDOWN(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction can be run during CICS shutdown.

XMXD_DISABLED
The transaction is disabled from running during CICS shutdown.

XMXD_ENABLED
The transaction is enabled to run during CICS shutdown.

SPURGE(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction is defined as system-purgeable.

XMXD_NO
The transaction is not system-purgeable.

XMXD_YES
The transaction is system-purgeable.

STATUS(name1)
returns, in a 1-byte location (*name1*), an equated value indicating the status of the transaction definition.

XMXD_DISABLED
The transaction definition is disabled.

XMXD_ENABLED
The transaction definition is enabled.

STORAGE_CLEAR(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether task-lifetime storage, of tasks associated with this transaction definition, is to be cleared before it is freed by a FREEMAIN command.

XMXD_NO
Task-lifetime storage need not be cleared before it's freed.

XMXD_YES
Task-lifetime storage must be cleared before it's freed.

STORAGE_FREEZE(name1 | (Rn))
returns, in a 1-byte location (*name1*), an equated value indicating whether storage freeze is defined for the transaction by means of the STGFRZ option on the CICS-supplied field engineering transaction, CSFE.

XMXD_NO
Storage is freed normally during the running of the transaction.

XMXD_YES
Storage that is normally freed during the running of a transaction is frozen.

SYSTEM_ATTACH(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the tasks attached with this tranid are always to be attached as system tasks.

XMXD_NO
A user task is being attached for this transaction.

XMXD_YES
A system task is being attached for this transaction.

SYSTEM_RUNAWAY(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction definition specifies the system default runaway-task time limit, which is specified on the **ICVR** system initialization parameter.

XMXD_NO
The transaction is not governed by the system runaway limit.

XMXD_YES
The transaction definition specifies the system default runaway limit.

TASKDATAKEY(name1)
returns, in a 1-byte location (*name1*), an equated value indicating the storage key of task-lifetime storage for tasks associated with this transaction definition.

XMXD_CICS
CICS key is specified for task-lifetime storage.

XMXD_USER
USER key is specified for task-lifetime storage.

TASKDATALOC(name1)
returns, in a 1-byte location (*name1*), an equated value indicating the data location of task-lifetime storage for tasks associated with this transaction definition.

XMXD_ANY
Task-lifetime storage can be located above 16 MB in virtual storage.

XMXD_BELOW
Task-lifetime storage must be located below 16 MB in virtual storage.

TCLASS(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction belongs to a transaction class.

XMXD_NO
The transaction is not a member of a transaction class.

XMXD_YES
The transaction is a member of the transaction class named in the TCLASS_NAME parameter.

TCLASS_NAME(name8)
returns the name of the transaction class to which the transaction belongs.

name8
The name of an 8-byte location to receive transaction class name to which the transaction belongs.

TPURGE(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction is defined as purgeable in the event of a z/OS Communications Server terminal error.

XMXD_NO
The transaction can not be purged if a terminal error occurs.

XMXD_YES
The transaction can be purged if a terminal error occurs.

TRACE(name1)
returns, in a 1-byte location (*name1*), an equated value indicating the level of tracing defined for the transaction:

XMXD_SPECIAL
CICS special-level trace This is the result of special trace being set by means of an EXEC CICS SET TRANSACTION command.

XMXD_STANDARD
CICS standard-level trace This equates to TRACE(YES) in the TRANSACTION resource definition.

XMXD_SUPPRESSED
Tracing is suppressed for the transaction This equates to TRACE(NO) in the TRANSACTION resource definition.

TRAN_PRIORITY(name4 | (Rn))
returns the transaction priority specified on the transaction definition.

name4
The name of a 4-byte location to receive the transaction priority, expressed as a binary value.

(Rn)
A register to receive the transaction priority, expressed as a binary value.

TRAN_ROUTING_PROFILE(name8)
returns the name of the profile that CICS is to use to route the transaction to a remote system.

name8
The name of an 8-byte location to receive the transaction-routing profile.

TRANSACTION_ID(name4)
returns the primary transaction identifier for this transaction definition inquiry.

name4
The name of a 4-byte location that contains the name of the transaction identifier.

TWASIZE(name4 | (Rn))
returns the size of the transaction work area specified on the transaction definition.

name4
The name of a 4-byte location to receive the size of the transaction work area, expressed as a binary value.

(Rn)
A register to receive the size of the transaction work area, expressed as a binary value.

RESPONSE and REASON values for INQUIRE_TRANDEF

RESPONSE	***REASON***
OK	None
EXCEPTION	UNKNOWN_TRANSACTION_ID
INVALID	None
DISASTER	LOGIC_ERROR
PURGED	None

The INQUIRE_TRANSACTION call

The INQUIRE_TRANSACTION function is provided on the DFHXMIQX macro call. Its purpose is to allow you to obtain information about a transaction that is attached (task). In general, this call is equivalent to the EXEC CICS INQUIRE TASK command, with some minor differences.

INQUIRE_TRANSACTION

```
DFHXMIQX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(INQUIRE_TRANSACTION),
       [TRANSACTION_TOKEN(name8),]]
       [OUT,
       [ATTACH_TIME(name8),]
       [CICS_UOW_ID(name8),]
       [DTIMEOUT(name4 | (Rn)),]
       [DYNAMIC(name1),]
       [FACILITY_NAME(name4),]
       [FACILITY_TYPE(name1),]
       [INITIAL_PROGRAM(name8),]
       [NETNAME(name8),]
       [ORIGINAL_TRANSACTION_ID(name4),]
       [OUT_TRANSACTION_TOKEN(name8),]
       [RE_ATTACHED_TRANSACTION(name1),]
       [REMOTE(name1),]
       [REMOTE_NAME(name8),]
       [REMOTE_SYSTEM(name4),]
       [RESOURCE_NAME(name16),]
       [RESOURCE_TYPE(name8),]
       [RESTART(name1),]
       [RESTART_COUNT(name2 | (Rn)),]
       [SPURGE(name1),]
       [START_CODE(name1),]
       [STATUS(name1),]
       [SUSPEND_TIME(name4 | (Rn)),]
       [SYSTEM_TRANSACTION(name1),]
       [TASK_PRIORITY(name1),]
       [TCLASS(name1),[TCLASS_NAME(name8),]]
       [TERMINATE_PROTECTED(name1),]       [TPURGE(name1),]
       [TRANNUM(name4 | string | ‘string'),]
       [TRAN_PRIORITY(name1),]
       [TRAN_ROUTING_PROFILE(name8),]
       [TRANSACTION_ID(name4),]
       [USERID(name8),]
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

The descriptions of the following parameters are the same as the corresponding parameters on the INQUIRE_TRANDEF function call.

```
DTIMEOUT
DYNAMIC
INITIAL_PROGRAM
REMOTE
REMOTE_NAME
REMOTE_SYSTEM
RESTART
SPURGE
STATUS
TCLASS
TRAN_ROUTING_PROFILE
TRANSACTION_ID
```

The parameter descriptions that follow explain briefly the possible values that can be returned on an INQUIRE_TRANSACTION call. For a more detailed explanation of these parameters, see the corresponding parameter descriptions for the TRANSACTION resource definition in TRANSACTION attributes.

ATTACH_TIME(name8)
returns the time in milliseconds since the task was attached.

name8
The name of an 8-byte location to receive the time, in packed decimal ABSTIME format.

CICS_UOW_ID(name8)
returns the CICS unit of work identifier for the task.

name8
The name of an 8-byte location to receive the unit of work id.

FACILITY_NAME(name4)
returns the name of the principal facility associated with the task.

name4
The name of a 4-byte location to receive the name of the principal facility.

FACILITY_TYPE(name1)
returns, in a 1-byte location (*name1*), an equated value indicating the type of principal facility associated with the task.

XMIQ_NONE
There is no principal facility.

XMIQ_START
The principal facility is an interval control element.

XMIQ_TD
The principal facility is a transient data queue.

XMIQ_TERMINAL
The principal facility is a terminal.

NETNAME(name8)
returns the network name of the principal facility associated with this task.

name8
The name of an 8-byte location to receive the network name.

ORIGINAL_TRANSACTION_ID(name4)
returns the transaction id that was used to attach the transaction. For example, if an alias was used at a terminal, this field returns the alias.

name4
The name of a 4-byte location to receive the name of the original transaction identifier.

OUT_TRANSACTION_TOKEN(name8)
returns the token that represents the task.

name8
The name of an 8-byte location to receive the transaction token for the task.

RE_ATTACHED_TRANSACTION(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the transaction has been re-attached.

XMIQ_NO
The transaction has not been re-attached and the global user exit program is invoked in the same environment as the original transaction-attach.

XMIQ_YES
The transaction has been re-attached and the global user exit program is invoked in a different environment from the original transaction-attach.

RESOURCE_NAME(name16)
returns the name of a resource that the (suspended) transaction waiting for.

name16
The name of an 16-byte location to receive the name of the resource on which the transaction is waiting.

RESOURCE_TYPE(name8)
returns the type of resource that the (suspended) transaction waiting for.

name8
The name of an 8-byte location to receive the type of resource on which the transaction is waiting.

RESTART_COUNT(name2 | (Rn))
returns the number of times this instance of the transaction has been restarted.

name2
The name of a 2-byte location to receive the number of times the transaction has been restarted, expressed as a half-word binary value.

(Rn)
A register to receive the number of times the transaction has been restarted, expressed as a half-word binary value.

START_CODE(name1)
returns, in a 1-byte location (*name1*), an equated value indicating how the task was started:

C
A CICS internal attach.

XMIQ_DF
The start code isn't yet known—to be set later.

XMIQ_QD
A transient data trigger level attach.

XMIQ_S
A START command without any data.

XMIQ_SD
A START command with data.

XMIQ_SZ
A front end programming interface (FEPI) attach.

XMIQ_T
A terminal input attach.

XMIQ_TT
A permanent transaction terminal attach.

SUSPEND_TIME(name4 | (Rn))
returns the length of time that the task has been in its current suspended state.

name4
The name of a 4-byte location to receive the number of seconds, rounded down, the task has been suspended, expressed as a binary value.

(Rn)
A register to receive the number of seconds, rounded down, the task has been suspended, expressed as a binary value.

SYSTEM_TRANSACTION(name1)
returns, in a 1-byte location (*name1*), an equated value indicating whether the task is CICS system task.

XMIQ_NO
The task is not a CICS system task.

XMIQ_YES
The task is a CICS system task.

TASK_PRIORITY(name1)
returns the combined task priority, which is the sum of the priorities defined for the terminal, transaction, and operator.

name1
The name of a 1-byte location to receive the task priority, expressed as a binary number.

TERMINATE_PROTECTED(name1)
returns, in a 1-byte location (name1), an equated value indicating whether the transaction can be killed.

XMIQ_NO
The transaction can be killed.

XMIQ_YES
The transaction cannot be killed.

TRANNUM(name4)
returns the task number of the transaction.

name4
The name of a 4-byte location to receive the task number.

TRANSACTION_TOKEN(name8)
specifies the transaction token for the task being inquired upon. This parameter is optional, and if omitted, the current task is assumed.

If you issue this call within an XXMATT global user exit program, the current task may be a CICS system task. To inquire on the user task for which XXMATT is invoked, you must specify the transaction token passed on the XXMATT exit-specific parameter list.

name8
The name of an 8-byte location that contains the transaction token.

USERID(name8)
returns the userid associated with this task.

name8
The name of an 8-byte location to receive the userid.

RESPONSE and REASON values for INQUIRE_TRANSACTION

RESPONSE	*REASON*
OK	None
DISASTER	ABEND
	LOOP
INVALID	None
EXCEPTION	NO_TRANSACTION_ENVIRONMENT
	BUFFER_TOO_SMALL
	INVALID_TRANSACTION_TOKEN
KERNERROR	None

The SET_TRANSACTION call

The SET_TRANSACTION function is provided on the DFHXMIQX macro call. Its purpose is to allow you to change the task priority and transaction class of the current task.

Note that you can use this call to change the TCLASS_NAME only when it is invoked from an XXMATT global user exit program.

SET_TRANSACTION

```
DFHXMIQX  [CALL,]
       [CLEAR,]
       [IN,
       FUNCTION(SET_TRANSACTION),
       [TASK_PRIORITY(name4),]
       [TCLASS_NAME(name8),]
```

```
       [TRANSACTION_TOKEN(name8),]]
       [OUT,
       RESPONSE (name1 | *),
       REASON (name1 | *)]
```

This command is threadsafe.

TASK_PRIORITY(name4)
specifies the new task priority being set for the task identified by TRANSACTION_TOKEN.

name4
The name of a 4-byte location that contains the new task priority number, expressed as a binary value.

TCLASS_NAME(name8)
specifies the new transaction class name that you want to associate this task with. To specify that the task is not to be in any transaction class, specify the special default system name DFHTCL00.

name8
The name of an 8-byte location that contains the name of the new transaction class. Set this field to DFHTCL00 for no transaction class.

TRANSACTION_TOKEN(name8)
specifies the transaction token that represents the task being modified. If you omit this parameter, the call defaults to the current task.

name8
The name of an 8-byte location that contains the transaction token.

RESPONSE and REASON values for SET_TRANSACTION

RESPONSE	***REASON***
OK	None
EXCEPTION	NO_TRANSACTION_ENVIRONMENT
	UNKNOWN_TCLASS
	INVALID_TRANSACTION_TOKEN
DISASTER	ABEND
	LOOP
INVALID	None
KERNERROR	None

Chapter 17. User journaling XPI function

The XPI provides one user journaling function, which is the DFHJCJCX call WRITE_JOURNAL_DATA.

Restriction: DFHJCJCX calls cannot be used in any exit program invoked from any global user exit point in the:

- Statistics domain
- Monitor domain
- Dump domain
- Dispatcher domain
- Transient data program.

The WRITE_JOURNAL_DATA call

WRITE_JOURNAL_DATA writes a single journal record to the journal specified in the journal model definition that matches the journal name (either a journal on an MVS system logger log stream, an SMF data set, or no record is written where DUMMY is defined in the definition).

WRITE_JOURNAL DATA

```
DFHJCJCX [CALL,]
      [CLEAR,]
      [IN,
      FUNCTION(WRITE_JOURNAL_DATA),
      FROM(block-descriptor),
      JOURNALNAME(name8 | string | 'string' ) |
      JOURNAL_RECORD_ID(name2 | string | 'string'),
      WAIT(YES|NO),
      [RECORD_PREFIX(block-descriptor),]]
      [OUT,
      RESPONSE(name1 | *),
      REASON(name1 | *)]
```

This command is threadsafe.

Important

There is a restriction in using the XPI early during initialization. Do not start exit programs that use the XPI functions TRANSACTION_DUMP, WRITE_JOURNAL_DATA, MONITOR, and INQUIRE_MONITOR_DATA until the second phase of the PLTPI. For further information about the PLTPI, refer to Writing initialization and shutdown programs .

FROM(block-descriptor)
specifies the address and the length of the journal record.

The block-descriptor comprises 8 bytes of data. The first 4 bytes hold the address of the data to be written. The second 4 bytes hold the length of the data. The block-descriptor is moved by the DFHJCJCX macro call to the location JCJC_FROM, which is mapped by the DFHJCJCY DSECT.

JOURNALNAME(name8 | string | "string")
specifies the name of the CICS journal or log to which the FROM data is to be written.

JOURNAL_RECORD_ID(name2 | string | "string")
specifies a 2-character value to be written to the journal record to identify its origin.

name2
The name of a 2-byte location

string
A character string that is limited to a length of 2 in the generated code

"string"
A character string enclosed in quotation marks, limited to a length of 2 in the generated code.

RECORD_PREFIX(block-descriptor)
specifies the optional user prefix.

WAIT(YES|NO)
specifies whether CICS is to wait until the record is written to the journal or log before returning control to the exit program.

RESPONSE and REASON values for WRITE_JOURNAL_DATA

RESPONSE	***REASON***
OK	None
EXCEPTION	IO_ERROR
	JOURNAL_NOT_FOUND
	JOURNAL_NOT_OPEN
	LENGTH_ERROR
	STATUS_ERROR
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in Making an XPI call.

Chapter 18. Threadsafe XPI commands

Most, but not all, XPI commands are threadsafe. Issuing any of the non-threadsafe commands causes CICS to use the QR TCB to ensure serialization.

The XPI commands that are threadsafe are indicated in the command syntax diagrams in with the statement: "This command is threadsafe", and are listed here.

Threadsafe commands

- DFHAPIQX INQ_APPLICATION_DATA
- DFHBRIQX INQUIRE_CONTEXT
- DFHDDAPX BIND_LDAP
- DFHDDAPX END_BROWSE_RESULTS
- DFHDDAPX FLUSH_LDAP_CACHE
- DFHDDAPX FREE_SEARCH_RESULTS
- DFHDDAPX GET_ATTRIBUTE_VALUE
- DFHDDAPX GET_NEXT_ATTRIBUTE
- DFHDDAPX GET_NEXT_ENTRY
- DFHDDAPX SEARCH_LDAP
- DFHDDAPX START_BROWSE_RESULTS
- DFHDDAPX UNBIND_LDAP
- DFHDSATX CHANGE_PRIORITY
- DFHDSSRX ADD_SUSPEND
- DFHDSSRX DELETE_SUSPEND
- DFHDSSRX RESUME
- DFHDSSRX SUSPEND
- DFHDSSRX WAIT_MVS
- DFHDUDUX SYSTEM_DUMP
- DFHJCJCX WRITE_JOURNAL_DATA
- DFHKEDSX START_PURGE_PROTECTION
- DFHKEDSX STOP_PURGE_PROTECTION
- DFHLDLDX ACQUIRE_PROGRAM
- DFHLDLDX DEFINE_PROGRAM
- DFHLDLDX DELETE_PROGRAM
- DFHLDLDX IDENTIFY_PROGRAM
- DFHLDLDX RELEASE_PROGRAM
- DFHLGPAX INQUIRE_PARAMETERS
- DFHLGPAX SET_PARAMETERS
- DFHMNMNX INQUIRE_MONITORING_DATA
- DFHMNMNX MONITOR
- DFHNQEDX DEQUEUE
- DFHNQEDX ENQUEUE
- DFHPGAQX INQUIRE_AUTOINSTALL
- DFHPGAQX SET_AUTOINSTALL

- DFHPGISX END_BROWSE_PROGRAM
- DFHPGISX GET_NEXT_PROGRAM
- DFHPGISX INQUIRE_CURRENT_PROGRAM
- DFHPGISX INQUIRE_PROGRAM
- DFHPGISX SET_PROGRAM
- DFHPGISX START_BROWSE_PROGRAM
- DFHSAIQX INQUIRE_SYSTEM
- DFHSAIQX SET_SYSTEM
- DFHSMMCX GETMAIN
- DFHSMMCX FREEMAIN
- DFHSMMCX INQUIRE_ELEMENT_LENGTH
- DFHSMMCX INQUIRE_TASK_STORAGE
- DFHSMSRX INQUIRE_ACCESS
- DFHSMSRX INQUIRE_SHORT_ON_STORAGE
- DFHSMSRX SWITCH_SUBSPACE
- DFHTRPTX TRACE_PUT
- DFHXMCLX INQUIRE_TCLASS
- DFHXMIQX INQUIRE_TRANSACTION
- DFHXMIQX SET_TRANSACTION
- DFHXMSRX INQUIRE_DTRTRAN
- DFHXMSRX INQUIRE_MXT
- DFHXMXDX INQUIRE_TRANDEF

Non-threadsafe commands

- DFHDUDUX TRANSACTION_DUMP

Notices

This information was developed for products and services offered in the U.S.A. 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 rights 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
United States of America

For license inquiries regarding double-byte (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 supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who want 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
United States of America

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

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

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.

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.

Programming interface information

CICS supplies some documentation that can be considered to be Programming Interfaces, and some documentation that cannot be considered to be a Programming Interface.

Programming Interfaces that allow the customer to write programs to obtain the services of CICS Transaction Server for z/OS, Version 5 Release 6 are included in the following sections of the online product documentation:

- Developing applications
- Developing system programs
- CICS TS security
- Developing for external interfaces
- Application development reference
- Reference: system programming
- Reference: connectivity

Information that is NOT intended to be used as a Programming Interface of CICS Transaction Server for z/OS, Version 5 Release 6 , but that might be misconstrued as Programming Interfaces, is included in the following sections of the online product documentation:

- Troubleshooting and support
- CICS TS diagnostics reference

If you access the CICS documentation in manuals in PDF format, Programming Interfaces that allow the customer to write programs to obtain the services of CICS Transaction Server for z/OS, Version 5 Release 6 are included in the following manuals:

- Application Programming Guide and Application Programming Reference
- Business Transaction Services
- Customization Guide

- C++ OO Class Libraries
- Debugging Tools Interfaces Reference
- Distributed Transaction Programming Guide
- External Interfaces Guide
- Front End Programming Interface Guide
- IMS Database Control Guide
- Installation Guide
- Security Guide
- Supplied Transactions
- CICSPlex® SM Managing Workloads
- CICSPlex SM Managing Resource Usage
- CICSPlex SM Application Programming Guide and Application Programming Reference
- Java™ Applications in CICS

If you access the CICS documentation in manuals in PDF format, information that is NOT intended to be used as a Programming Interface of CICS Transaction Server for z/OS, Version 5 Release 6 , but that might be misconstrued as Programming Interfaces, is included in the following manuals:

- Data Areas
- Diagnosis Reference
- Problem Determination Guide
- CICSPlex SM Problem Determination Guide

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 www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Apache, Apache Axis2, Apache Maven, Apache Ivy, the Apache Software Foundation (ASF) logo, and the ASF feather logo are trademarks of Apache Software Foundation.

Gradle and the Gradlephant logo are registered trademark of Gradle, Inc. and its subsidiaries in the United States and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Red Hat®, and Hibernate® are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Spring Boot is a trademark of Pivotal Software, Inc. in the U.S. and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Zowe™, the Zowe logo and the Open Mainframe Project™ are trademarks of The Linux Foundation.

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:

For the CICSPlex SM Web User Interface (main interface):
Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user's user name and other personally identifiable information for purposes of session management, authentication, enhanced user usability, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the CICSPlex SM Web User Interface (data interface):
Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's user name and other personally identifiable information for purposes of session management, authentication, or other usage tracking or functional purposes. These cookies cannot be disabled.

For the CICSPlex SM Web User Interface ("hello world" page):
Depending upon the configurations deployed, this Software Offering may use session cookies that collect no personally identifiable information. These cookies cannot be disabled.

For CICS Explorer®:
Depending upon the configurations deployed, this Software Offering may use session and persistent preferences that collect each user's user name and password, for purposes of session management,

authentication, and single sign-on configuration. These preferences cannot be disabled, although storing a user's password on disk in encrypted form can only be enabled by the user's explicit action to check a check box during sign-on.

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.

For more information about the use of various technologies, including cookies, for these purposes, see IBM Privacy Policy and IBM Online Privacy Statement, the section entitled *Cookies, Web Beacons and Other Technologies* and the IBM Software Products and Software-as-a-Service Privacy Statement.

Index

M

O

P

R

S

T

U

W

X