IBM

iSeries

Database and File APIs

Version 5 Release 3

iSeries

Database and File APIs

Version 5 Release 3

Note

Before using this information and the product it supports, be sure to read the information in "Notices," on page 577.

Sixth Edition (August 2005)

This edition applies to version 5, release 3, modification 0 of Operating System/400 (product number 5722-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

Contents

Database and File APIs

The database and file APIs retrieve specific information about OS/400(R) files. These APIs also have the ability to get data and manipulate files.

With the exception of QDBLDBR, QDFRTVFD, QSQPRCED, QDBSTRS, QDBSTDS, QDBSTUS, QDBSTCRS, QDBSTLS, QDBSTLDS, and QDBSTLRS, the database and file APIs work with files that are either local or remote. Local files are files that are on the server where the program is running. Remote files are files on a target (remote) system that are accessed using a distributed data management (DDM) file on a source (local) system. DDM files provide the information needed for a local system to locate a remote system and to access data in the remote system″s database files. The QDBLDBR, QDFRTVFD, QSQPRCED, QDBSTRS, QDBSTDS, QDBSTUS, QDBSTCRS, QDBSTLS, QDBSTLDS, and QDBSTLRS APIs work with local database files only.

When you call these APIs from a high-level language (HLL) program, you must specify whether to use file override processing on your local or remote files. The QDBLDBR, QSQPRCED, and QDMRTVFO APIs, however, do not support overrides.

Some of the database and file APIs return character values that have an associated coded character set identifier (CCSID). If the CCSID value for the job calling the API is not 65535, the character values are converted from their current CCSID to the CCSID of the job. This conversion may cause some data to be lost. The CCSID associated with the job is returned to the user. If the CCSID value for the job is 65535, no conversions are performed on the character values. The character value CCSID stored in the file object is returned to the user.

The database and file APIs use the standard user space format for the lists of information they return. If you are not familiar with this format, see User Space Format for List APIs before using these APIs.

The database and file APIs include:
- "File APIs" on page 2
- "Database APIs" on page 259

A database exit program provides additional (user-written) functions for the database. The database and file exit programs are:
- "CLI Connection Exit Program" on page 529 exit program is called by CLI through the registration facility before the connection is made to the relational database.
- "Close Database File Exit Program" on page 530 exit program is called when a process is trying to lock a file that is held by another process.
- "Open Database File Exit Program" on page 531 exit program is called when a when a job is opening a database file.
- "SQL Client Integration Exit Program" on page 535 exit program enables SQL applications to access data managed by a database management system other than the OS/400 relational database.

Top | APIs by category

APIs

These are the APIs for this category.

File APIs

The file APIs are:

- “List Database File Members (QUSLMBR) API” on page 3 (QUSLMBR) generates a list of database file members and places the list in a user space.
- “List Database Relations (QDBLDBR) API” on page 9 (QDBLDBR) provides information on how files and members are related to a specified database file.
- “List Fields (QUSLFLD) API” on page 15 (QUSLFLD) generates a list of fields within a specified file record format name.
- “List Open Files (QDMLOPNF)” on page 29 (QDMLOPNF) generates a list of *FILE objects that currently are open in the job or that were opened by the thread that is specified in the job identification information input parameter.
- “List Record Formats (QUSLRCD) API” on page 35 (QUSLRCD) generates a list of record formats contained within a specified file.
- » “Replay Database Operation (QDBRPLAY) API” on page 41 (QDBRPLAY) replays a database operation from a single journal entry.«
- “Retrieve Database File Description (QDBRTVFD) API” on page 48 (QDBRTVFD) provides complete and specific information about a file on a local or remote system.
- “Retrieve Display File Description (QDFRTVFD) API” on page 133 (QDFRTVFD) allows you to get specific information about the data description specifications (DDS) definition used to create a display file.
- “Retrieve File Override Information (QDMRTVFO) API” on page 212 (QDMRTVFO) returns the name of the file, library, member, and final type of override that result from processing overrides for a specified file name.
- “Retrieve Job Record Locks (QDBRJBRL) API” on page 215 (QDBRJBRL) lets you generate a list of record locks that a specific job or thread is holding or for which it is waiting.
- “Retrieve Member Description (QUSRMBRD) API” on page 224 (QUSRMBRD) returns specific information about a single database file member.
- “Retrieve Record Locks (QDBRRCDL) API” on page 249 (QDBRRCDL) lets you generate a list of jobs that are either waiting for or holding a specific record lock.
- “Retrieve Short Name (QDBRTVSN) API” on page 258 (QDBRTVSN) allows you to get the 10-character name of a database file by requesting the long file name of the database file.

«

Top | Database and File APIs | APIs by category

List Database File Members (QUSLMBR) API

Required Parameter Group:

1	Qualified user space name	
Input	Char(20)	
2	Format name	
Input	Char(8)	
3	Qualified database file name	
Input	Char(20)	
4	Member name	
Input	Char(10)	
5	Override processing	
Input	Char(1)	

Optional Parameter:

6	Error code
I/O	Char(*)

Service Program Name: QUSLMBR

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 8.

The List Database File Members (QUSLMBR) API generates a list of database file members and places the list in a specified user space. When you specify a generic member name, you can generate a subset of the member list. You can use the QUSLMBR API with database file types *PF, *LF, and *DDMF. The generated list replaces any existing information in the user space. The file members listed in the user space are not in any predictable order. To retrieve additional information about each member in the list, see the "Retrieve Member Description (QUSRMBRD) API" on page 224.

You can use the QUSLMBR API to:
- List members more quickly than by using the *MBRLIST value on the TYPE parameter of the Display File Description (DSPFD) command.
- Retrieve information for all of the members of a database file more quickly and easily than by multiple calls to the Retrieve Member Description (QUSRMBRD) API. It is your discretion to decide which API best suits the needs of your application. For example, if you want to selectively retrieve member descriptions for a subset of the member list, you might want to use both the QUSLMBR and QUSRMBRD APIs.
- Ensure that the last date the source was changed matches the date of the source used to create the object.

Authorities and Locks

User Space Authority
*CHANGE

User Space Library Authority
: *EXECUTE

File Authority
: *OBJOPR

User Space Lock
: *EXCLRD

File Lock
: *SHRRD

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)

The user space that is to receive the created list. The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. You can use these special values for the library name:

**CURLIB*	The job's current library
**LIBL*	The library list

Format name
INPUT; CHAR(8)

The content and format of the information returned for each member. The possible format names are:

MBRL0100	Member name
MBRL0200	Member name and source information This format requires more processing than the MBRL0100 format.
MBRL0310	Member name and basic description. The member information is the same as that generated by the Retrieve Member Description (QUSRMBRD) API using format MBRD0100. This format requires more system processing and takes longer to produce than the MBRL0200 format.
MBRL0320	Member name and expanded description. The member information is the same as that generated by the Retrieve Member Description (QUSRMBRD) API using format MBRD0200. The additional information requires more system processing and takes longer to produce than the MBRL0310 format.
MBRL0330	Member name and full description. The member information is the same as that generated by the Retrieve Member Description (QUSRMBRD) API using format MBRD0300. The additional information requires more system processing and takes longer to produce than the MBRL0320 format.

For more information, see "MBRL0100 List Data Section" on page 6, "MBRL0200 List Data Section" on page 6, or "MBRL0300 List Data Section" on page 7.

Qualified database file name
INPUT; CHAR(20)

The name of the database file whose member names are to be placed in the list. The first 10 characters contain the database file name, and the second 10 characters contain the name of the library where the file is located. You can use these special values for the library name:

**CURLIB*	The job's current library
**LIBL*	The library list

Member name
INPUT; CHAR(10)

A specific member name, a generic member name, or this special value:

**ALL*	All members

Override processing
INPUT; CHAR(1)

Whether overrides are to be processed. The following character values are used:

0	No override processing
1	Override processing

Optional Parameter

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Format of the Generated Lists

The file member list consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- A list data section:
  - MBRL0100 format
  - MBRL0200 format
  - MBRL0300 format

The MBRL0300 list data section is generated if MBRL0310, MBRL0320, or MBRL0330 is specified as the format name parameter.

For details about the user area and generic header, see User Space Format for List APIs. For details about the remaining items, see the following sections. For detailed descriptions of the fields in the list returned, see "Field Descriptions" on page 7.

When you retrieve list entry information from a user space, you must use the entry size returned in the generic header as a displacement to the next list entry. The size of each entry may be padded at the end. If you do not use the entry size, the result may not be valid. For examples of how to process lists, see Examples.

Input Parameter Section

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	File name specified

Offset			
Dec	Hex	Type	Field
38	26	CHAR(10)	File library name specified
48	30	CHAR(10)	Member name specified
58	3A	CHAR(1)	Override processing

Header Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	File name used
10	A	CHAR(10)	File library name used
20	14	CHAR(10)	File attribute
30	1E	CHAR(50)	File text description
80	50	BINARY(4)	Total number of members in file
84	54	CHAR(1)	Source file
85	55	CHAR(3)	Reserved
88	58	BINARY(4)	File text description CCSID

MBRL0100 List Data Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Member name used

MBRL0200 List Data Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Member name used
10	A	CHAR(10)	Source type
20	14	CHAR(13)	Creation date and time
33	21	CHAR(13)	» Last source change or table refresh date and time «
46	2E	CHAR(50)	Member text description
96	60	BINARY(4)	Member text description CCSID

MBRL0300 List Data Section

The MBRL0300 format provides an offset to a retrieved member description. The member descriptions have the same format as those generated by the Retrieve Member Description (QUSRMBRD) API. The following relationship exists between the QUSLMBR format name and the QUSRMBRD member descriptions:

- MBRL0310 generates an MBRD0100 description.
- MBRL0320 generates an MBRD0200 description.
- MBRL0330 generates an MBRD0300 description.

For more information about the member description formats, see "Retrieve Member Description (QUSRMBRD) API" on page 224.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Member name used
10	A	CHAR(2)	Reserved
12	C	BINARY(4)	Offset to member description information
16	10	CHAR(16)	Reserved

Field Descriptions

Creation date and time. The date and time the member was created. The format of this field is in the CYYMMDDHHMMSS as follows:

C	Century, where 0 indicates years 19*xx* and 1 indicates years 20*xx*.
YY	Year
MM	Month
DD	Day
HH	Hour
MM	Minute
SS	Second

File attribute. The type of file found:

PF	Physical file
LF	Logical file
DDMF	Distributed data management file

File library name specified. The name of the library containing the file whose member names are to be placed in the list.

File library name used. The name of the library containing the file whose member names are placed in the list.

File name specified. The name of the file specified in the call to the API.

File name used. The name of the file whose member names are placed in the list.

File text description. The description of the file.

File text description CCSID. The CCSID for the file text description. The job default CCSID of the current process will be used to translate the text. For more information about CCSID, see the Globalization topic.

Format name. The content and format of the information returned for each member.

» Last source change or table refresh date and time. For source files, the date and time that this source member was last changed. For SQL materialized query tables, the date and time that the last SQL Refresh Table statement refreshed this member. If the member has never been refreshed, this field will contain hexadecimal zeroes. « This field is in the CYYMMDDHHMMSS format where the format is the same as for the creation date and time field.

Member name specified. The name of the member specified in the call to the API.

Member name used. The name of a member found in the file.

Member text description. Description of the member found in the file.

Member text description CCSID. The CCSID for the member text description. The job default CCSID of the current process will be used to translate the text. For more information about CCSID, see the Globalization topic.

Offset to member description information. The number of bytes from the beginning of the user space to the beginning of the retrieved member description.

Override processing. Whether overrides are to be processed. The possible values are:

0	No override processing
1	Override processing

Reserved. An ignored field.

Source file. Whether the file is a source file or a data file. The possible values are:

0	Data file
1	Source file

Source type. The type of source member if this is a source file. Some possible values are:
- CL
- COBOL
- RPG
- TXT

Total number of members in file. The total number of members in the file specified.

User space library name. The name of the library that contains the user space that is to receive the generated list.

User space name. The name of the user space that is to receive the generated list.

Usage Notes

In multithreaded jobs, this API is not threadsafe and fails for distributed data management (DDM) files of type *SNA.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3CF1 E	Error code parameter not valid.
CPF3C20 E	Error found by program &1.
CPF3C21 E	Format name &1 is not valid.
CPF3C22 E	Cannot get information about file &1.
CPF3C23 E	Object &1 is not a database file.
CPF3C25 E	Value &1 for file override parameter is not valid.
CPF3C27 E	Cannot get information about member &3 from file &1.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C90 E	Literal value cannot be changed.
CPF8100 E	All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9800 E	All CPF98xx messages could be signaled. xx is from 01 to FF.

API introduced: V1R3

Top | "Database and File APIs," on page 1 | APIs by category

List Database Relations (QDBLDBR) API

Required Parameter Group:

1	Qualified user space name
Input	Char(20)
2	Format
Input	Char(8)
3	Qualified file name
Input	Char(20)
4	Member
Input	Char(10)
5	Record format
Input	Char(10)
6	Error code
I/O	Char(*)

Default Public Authority: *USE

Threadsafe: No

The List Database Relations (QDBLDBR) API gives relational information about database files. The information identifies the physical and logical files that are dependent on a specific file, files that use a specific record format, or file members that are dependent on a specific file member. The information is placed in a user space specified by you.

Similar in function to the Display Database Relations (DSPDBR) command, this API allows more input parameter values than does the command. Also, your program can have more direct access to the information put in the user space by this API than when the command places similar information in an output file.

The information generated by this API replaces any existing information in the user space. It does not append information to any information already in the user space. If the space is bigger than needed, the contents of the remainder of the space are not changed. If the space is not big enough, it is extended.

Authorities and Locks

User Space Authority
 *CHANGE

User Space Library Authority
 *EXECUTE

User Space Lock
 *EXCLRD

File Authority
 *USE

File Library Authority
 *USE

File Lock
 *SHRNUPD

Required Parameter Group

Qualified user space name
 INPUT; CHAR(20)

The user space that is to receive the database relations information. The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. You can use these special values for the library name:

**CURLIB*	The job's current library
**LIBL*	The library list

Format
 INPUT; CHAR(8)

The content and format of the information to be returned about the specified file, member, or record format. One of the following format names must be used:

DBRL0100	File information
DBRL0200	Member information
DBRL0300	Record format information

For more information, see "DBRL0100 Format (File)" on page 12, "DBRL0200 Format (Member)" on page 12, or "DBRL0300 Format (Record Format)" on page 13.

Qualified file name
 INPUT; CHAR(20)

The name of the file for which database relations information is to be extracted. The first 10 characters contain the file name, and the second 10 characters contain the name of the library where the file is located. The file name cannot be a DDM file. The file name can be a specific file name, a generic name, or the following special value:

**ALL*	All files

You can use these special values for the library name:

**ALL*	All libraries in the system
**ALLUSR*	All nonsystem libraries. For information on the libraries included, see *ALLUSR in Generic library names.
**CURLIB*	The job's current library
**LIBL*	The library list
**USRLIBL*	Libraries listed in the user portion of the library list

Member
INPUT; CHAR(10)

The name of the member to be used for retrieving database relations for format DBRL0200. This value can be a specific member name, a generic member name, or one of the following special values:

**FIRST*	Information about the first member (in the order created) in the specified file or files is to be provided.
**LAST*	Information about the last member (in the order created) in the specified file or files is to be provided.
**ALL*	Information about all members in the specified files is to be provided.

This parameter is ignored for formats DBRL0100 and DBRL0300.

Record format
INPUT; CHAR(10)

The name of the record format to be used for retrieving database relations for format DBRL0300. This value can be a specific record format, a generic record format, or the following special value:

**ALL*	All record formats in the specified file

This input is ignored for formats DBRL0100 and DBRL0200.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Format of the Generated List

The database relations list consists of an input parameter section and one of three possible formats for the list data section. The three formats are determined by the kind of information you are looking for. The format names are:

DBRL0100	Database relations (file)
DBRL0200	Database relations (member)
DBRL0300	Database relations (record format)

The layout of the contents of the user space is determined by the format used. The following tables show how the contents of the input parameter section and the data format sections are organized. For descriptions of each field, see "Field Descriptions" on page 13.

Input Parameter Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	File name specified
38	26	CHAR(10)	File library name specified
48	30	CHAR(10)	Member name specified
58	3A	CHAR(10)	Record format name specified

DBRL0100 Format (File)

The structure of the information returned is determined by the value specified for the format name. The DBRL0100 format includes information on files dependent on the file specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 13.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	File name used
10	A	CHAR(10)	File library name used
20	14	CHAR(10)	Dependent file name
30	1E	CHAR(10)	Dependent library name
40	28	CHAR(1)	Dependency type
41	29	CHAR(3)	Reserved
44	2C	BINARY(4)	Join reference number
48	30	CHAR(10)	Constraint library name
58	3A	BINARY(4)	Constraint name length
62	3E	CHAR(258)	Constraint name

DBRL0200 Format (Member)

The structure of the information returned is determined by the value specified for the format name. The DBRL0200 format includes information on files and members dependent on the file member specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 13.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	File name used
10	A	CHAR(10)	File library name used
20	14	CHAR(10)	Member name used
30	1E	CHAR(10)	Dependent file name

Offset			
Dec	Hex	Type	Field
40	28	CHAR(10)	Dependent library name
50	32	CHAR(10)	Dependent member name
60	3C	CHAR(1)	Dependency type
61	3D	CHAR(3)	Reserved
64	40	BINARY(4)	Join reference number
68	44	BINARY(4)	Join file number
72	48	CHAR(10)	Constraint library name
82	52	BINARY(4)	Constraint name length
86	56	CHAR(258)	Constraint name

DBRL0300 Format (Record Format)

The structure of the information returned is determined by the value specified for the format name. The DBRL0300 format includes information on files dependent on the record format specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	File name used
10	A	CHAR(10)	File library name used
20	14	CHAR(10)	Record format name used
30	1E	CHAR(10)	Dependent file name
40	28	CHAR(10)	Dependent library name

Field Descriptions

Constraint library name. The name of the library containing the file to which the constraint applies.

Constraint name. The name of the constraint. This only applies when the dependency type is *C*.

Constraint name length. The length of the constraint name. Delimited names can be a maximum of 258 characters and non-delimited names a maximum of 128 characters.

Dependency type. How a file or member is related to the file or member specified with the QDBLDBR API. Possible values are:

blank	No dependent files or members were found for the specified file.
C	Constraint.
D	The dependent file or member is dependent on the data in the specified file or member that was extracted.
I	The dependent file member is sharing the access path of the file that the information was extracted from.
O	If an access path is shared, one of the file members is considered the owner. The owner of the access path is charged with the storage used for the access path. If the member displayed is designated the owner, one or more file members are designated with an I for access path sharing.

V	The SQL view or member is dependent on the specified SQL view.

Dependent file name. The name of the file that is dependent on the file specified using the QDBLDBR API. If no dependent files are found for the file specified, the dependent file name is *NONE.

Dependent library name. The name of the library that the dependent file is in. If there are no dependent files found for the file specified, the dependent library name is blank.

Dependent member name. The name of the file member that is dependent on the file member specified using the QDBLDBR API. If no dependent members are found for the member specified, the dependent member name is *NONE.

File library name specified. The name of the library containing the file for which the database relations information is requested.

File library name used. The name of the library containing the file used to extract the database relations information in this list entry.

File name specified. The name of the file for which the database relations information is to be extracted.

File name used. The name of the file used to extract the database relations information in this list entry.

Format name. The name of the format in which the database relations information is returned to the user space.

Join file number. If the file for which database relations information is being extracted is a join logical file, this is the ordinal number of the file in the JFILE to which the dependency relates. The join file number is zero if either of the following are correct:
- No dependent files are found for the file specified.
- The file for which the information is being extracted is not a join file.

Join reference number. If the dependent file listed is a join logical file, this is the ordinal number of the file in the JFILE to which this dependency relates. The join reference number is zero if either of the following are correct:
- No dependent files are found for the file specified.
- The dependent file is not a join file.

Member name specified. The name of the member for which the information is extracted.

Member name used. The name of the member used to extract the database relations information in this list entry.

Record format name specified. The name of the record format for which the information is displayed.

Record format name used. The name of the record format used to extract the database relations information in this list entry.

Reserved. An ignored field.

User space library name. The name of the library that contains the user space that receives the database relations information requested.

User space name. The name of the user space that receives the database relations information requested.

Error Messages

Message ID	Error Message Text
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF3C21 E	Format name &1 is not valid.
CPF3C23 E	Object &1 is not a database file.
CPF3C90 E	Literal value cannot be changed.
CPF326C E	File name &1 not valid special value.
CPF326D E	Member name &1 not valid special value.
CPF326E E	Record format name &1 not valid special value.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R2

Top | "Database and File APIs," on page 1 | APIs by category

List Fields (QUSLFLD) API

Required Parameter Group:

1	Qualified user space name
Input	Char(20)
2	Format name
Input	Char(8)
3	Qualified file name
Input	Char(20)
4	Record format name
Input	Char(10)
5	Override processing
Input	Char(1)

Optional Parameter:

6	Error code
I/O	Char(*)

Default Public Authority: *USE

Threadsafe: No

The List Fields (QUSLFLD) API generates a list of fields within a specified file record format name. The list of fields is placed in a specified user space. The generated list replaces any existing information in the user space. You can use the QUSLFLD API only with database file types, such as *PF, *LF, and *DDMF, and device file types, such as *ICFF and *PRTF.

You can use the QUSLFLD API to:
- Generate a list of field format names.
- Gather additional information about specific field formats.

- Create a product similar to the Structured Query Language (SQL) using the Open Query File (OPNQRYF) command.
- Create applications similar to the data file utility (DFU).
- Create a compiler supporting externally described data.
- Create applications that use data defined to the system.

Authorities and Locks

User Space Authority
*CHANGE

User Space Library Authority
*EXECUTE

File Library Authority
*USE

File Authority
*OBJOPR

User Space Lock
*EXCLRD

File Lock
*SHRRD

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)

The name of the user space that is to receive the created list, and the library in which it is located. The first 10 characters contain the user space name, and the second 10 characters contain the library name. You can use these special values for the library name:

**CURLIB*	The job's current library
**LIBL*	The library list

Format name
INPUT; CHAR(8)

The format of the information returned. You must use the following format name:

FLDL0100	Field information
FLDL0200	Field and default value information
FLDL0300	Field, alternative field name, and default value information

For more information, see "Format of the Generated List" on page 17.

Qualified file name
INPUT; CHAR(20)

The file whose member names are to be placed in the list, and the library in which it is located. The first 10 characters contain the file name, and the second 10 characters contain the library name. You can use these special values for the library name:

**CURLIB*	The job's current library
**LIBL*	The library list

Record format name
INPUT; CHAR(10)

The record format name whose fields are to be returned.

Override processing
INPUT; CHAR(1)

Whether overrides are to be processed. The possible values are:

0	No override processing
1	Override processing

Optional Parameter

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error code parameter. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Format of the Generated List

The field list consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- The FLDL0100, FLDL0200, or FLDL0300 list data section

For details about the user area and generic header, see User space format for list APIs. For details about the remaining items, see the following sections. For descriptions of each field in the list returned, see "Field Descriptions" on page 20.

When you retrieve list entry information from a user space for format FLDL0100, you must use the entry size returned in the generic header as a displacement to the next list entry. The size of each entry may be padded at the end. If you do not use the entry size, the result may not be valid.

When you retrieve list entry information from a user space for format FLDL0200 or FLDL0300, you must use the length provided at the beginning of format FLDL0200 or FLDL0300 as a displacement to the next list entry. If you do not use the length provided in FLDL0200 or FLDL0300, the result may not be valid.

For examples of how to process lists, see the API examples.

Input Parameter Section

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	File name specified
38	26	CHAR(10)	File library name specified

Offset			
Dec	Hex	Type	Field
48	30	CHAR(10)	Record format name specified
58	3A	CHAR(1)	Override processing

Header Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	File name used
10	A	CHAR(10)	File library name used
20	14	CHAR(10)	File type
30	1E	CHAR(10)	Record format name used
40	28	BINARY(4)	Record length
44	2C	CHAR(13)	Record format ID
57	39	CHAR(50)	Record text description
107	6B	CHAR(1)	Reserved
108	6C	BINARY(4)	Record text description CCSID
112	70	CHAR(1)	Variable length fields in format indicator
113	71	CHAR(1)	Graphic fields indicator
114	72	CHAR(1)	Date and time fields indicator
115	73	CHAR(1)	Null-capable fields indicator

FLDL0100 List Data Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Field name
10	A	CHAR(1)	Data type
11	B	CHAR(1)	Use
12	C	BINARY(4)	Output buffer position
16	10	BINARY(4)	Input buffer position
20	14	BINARY(4)	Field length in bytes
24	18	BINARY(4)	Digits
28	1C	BINARY(4)	Decimal position
32	20	CHAR(50)	Field text description
82	52	CHAR(2)	Edit code
84	54	BINARY(4)	Edit word length
88	58	CHAR(64)	Edit word
152	98	CHAR(20)	Column heading 1
172	AC	CHAR(20)	Column heading 2

Offset		Type	Field
Dec	Hex		
192	C0	CHAR(20)	Column heading 3
212	D4	CHAR(10)	Internal field name
222	DE	CHAR(30)	Alternative field name
252	FC	BINARY(4)	Length of alternative field name
256	100	BINARY(4)	Number of DBCS characters
260	104	CHAR(1)	Null values allowed
261	105	CHAR(1)	Host variable indicator
262	106	CHAR(4)	Date and time format
266	10A	CHAR(1)	Date and time separator
267	10B	CHAR(1)	Variable length field indicator (overlay for MI mapping)
268	10C	BINARY(4)	Field text description CCSID
272	110	BINARY(4)	Field data CCSID
276	114	BINARY(4)	Field column headings CCSID
280	118	BINARY(4)	Field edit words CCSID
284	11C	BINARY(4)	UCS-2 displayed field length
288	120	BINARY(4)	Field data encoding scheme
292	124	BINARY(4)	Maximum large object field length
296	128	BINARY(4)	Pad length for large object
300	12C	BINARY(4)	Length of user-defined type name
304	130	CHAR(128)	User-defined type name
432	1B0	CHAR(10)	User-defined type library name
442	1BA	CHAR(1)	Datalink link control
443	1BB	CHAR(1)	Datalink integrity
444	1BC	CHAR(1)	Datalink read permission
445	1BD	CHAR(1)	Datalink write permission
446	1BE	CHAR(1)	Datalink recovery
447	1BF	CHAR(1)	Datalink unlink control
448	1C0	BINARY(4)	Display or print row number
452	1C4	BINARY(4)	Display or print column number
456	1C8	CHAR(1)	ROWID column
457	1C9	CHAR(1)	Identity column
458	1CA	CHAR(1)	GENERATED BY
459	1CB	CHAR(1)	Identity column - CYCLE
460	1CC	DECIMAL(31,0)	Identity column - Original START WITH
476	1DC	DECIMAL(31,0)	Identity column - Current START WITH
492	1EC	BINARY(4)	Identity column - INCREMENT BY
496	1F0	DECIMAL(31,0)	Identity column - MINVALUE
512	200	DECIMAL(31,0)	Identity column - MAXVALUE
528	210	BINARY(4)	Identity column - CACHE
532	214	CHAR(1)	Identity column - ORDER

Offset			
Dec	Hex	Type	Field
533	215	CHAR(11)	Reserved

FLDL0200 List Data Section

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Length of FLDL0200 format
4	4	BINARY(4)	Displacement to default value
8	8	BINARY(4)	Length of default value
12	C		All fields defined by FLDL0100 format
*	*	CHAR(*)	Default value

FLDL0300 List Data Section

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Length of FLDL0300 format
4	4	BINARY(4)	Displacement to all fields defined by FLDL0100 format
8	8	BINARY(4)	Displacement to alternative field name
12	C	BINARY(4)	Displacement to default value
16	10	BINARY(4)	Length of default value
*	*		All fields defined by FLDL0100 format
*	*	CHAR(*)	Alternative field name (long)
*	*	CHAR(*)	Default value

Field Descriptions

Alternative field name. The alternative name of the field the entry describes. This is the DDS keyword ALIAS or a long column name in SQL. If the length of the alternative name is greater than 30, the Alternative field name will contain blanks. If the FLDL0300 format is used, the Alternative field name (long) will always contain the alternative name.

Alternative field name (long). The alternative name of the field the entry describes. This is the DDS keyword ALIAS or a long column name in SQL.

Column heading 1. The description of the first column heading for this field. It contains blanks if the heading is not defined.

Column heading 2. The description of the second column heading for this field. It contains blanks if the heading is not defined.

Column heading 3. The description of the third column heading for this field. It contains blanks if the heading is not defined.

Data type. The type of field:

A	Alphanumeric (character)
B	Binary
D	Digits only
E	Either DBCS or alphanumeric
F	Floating point
G	Graphic data type
H	Hexadecimal
I	Inhibit entry
J	Double-byte character set (DBCS) data only
L	Date
M	Numeric only
N	Numeric shift
O	(Open) Both DBCS and alphanumeric
P	Packed decimal
S	Zoned decimal
T	Time
W	Katakana
X	Alphabetic only (character)
Y	Numeric only
Z	Timestamp
1	Binary large object (BLOB)
2	Character large object (CLOB)
3	Graphic data large object (DBCLOB)
4	Datalink
» *5*	Binary character «

Datalink integrity. How the control of the file is handled. This value applies to datalink fields. A **datalink** is a field data type that is used to point to another object that contains the data for that field. If the datalink link control field is 0 (no link control), this field is not applicable. The possible values are:

0	All linked files are under control of the database.
1	All linked files are under selective database control if the server has the Datalink File Manager installed.

Datalink link control. Whether the file should be linked by the Datalink File Manager. The **Datalink File Manager** is a function that tracks which files are linked to a specific database file. This value applies to datalink fields. The possible values are:

0	No link control.
1	File link control.

Datalink read permission. The check that is done to read the file. This value applies to datalink fields. If the datalink link control field is 0 (no link control), this field is not applicable. The possible values are:

0	The database controls whether a user has read authority.
1	The file system controls whether a user has read authority.

Datalink recovery. Whether file recovery is done. This value applies to datalink fields. If the datalink link control field is 0 (no link control), this field is not applicable. The possible values are:

0	Recovery is not done.
1	Recovery is done.

Datalink unlink control. The action that is done to a file during an unlink operation. This value applies to datalink fields. If the datalink link control field is 0 (no link control) or the datalink write permission field is 1 (File system control), this field is not applicable. The possible values are:

0	Restore the file owner and file authorities that existed prior to the file link when an unlink operation occurs.
1	Delete the file when an unlink operation occurs.

Datalink write permission. The check that is done to write to the file. This value applies to datalink fields. If the datalink link control field is 0 (no link control), this field is not applicable. The possible values are:

0	The file is blocked from accepting writing.
1	The file system controls whether a user has write authority.

Date and time fields indicator. Whether this format contains date and time fields. The possible values are:

0	The format does not contain date and time fields.
1	The format contains date and time fields.

Date and time format. This value applies to date, time, and timestamp fields. It also may apply to packed decimal, zoned decimal, and character fields in a logical file. The possible values are:

**USA*	IBM USA standard (mm/dd/yyyy, hh:mm a.m., hh:mm p.m.)
**ISO*	International Standards Organization (yyyy-mm-dd, hh.mm.ss)
**EUR*	IBM European Standard (dd.mm.yyyy, hh.mm.ss)
**JIS*	Japanese Industrial Standard Christian Era (yyyy-mm-dd, hh:mm:ss)
**SAA*	SAA timestamp
**MDY*	Month/day/year (mm/DD/yy)
**DMY*	Day/month/year (DD/mm/yy)
**YMD*	Year/month/day (yy/mm/DD)
**JUL*	Julian (yy/ddd)
**HMS*	Hour/minute/second (hh:mm:Ss)
MDYY	Month/day/year (mm/DD/yyyy)
DMYY	Day/month/year (DD/mm/yyyy)
YYMD	Year/month/day (yyyy/mm/DD)
JUL4	Long Julian (yyyy/ddd)
CMDY	Century/month/day/year (c/mm/DD/yy)
CDMY	Century/day/month/year (c/DD/mm/yy)
CYMD	Century/year/month/day (c/yy/mm/DD)
**MY*	Month/year (mm/yy)
**YM*	Year/month (yy/mm)
**MYY*	Month/year (mm/yyyy)
**YYM*	Year/month (yyyy/mm)

Date and time separator. This value applies only to date or time fields. The possible values are:

/	Slash separator
-	Dash separator
.	Period separator
,	Comma Separator

:	Colon separator
(blank)	Blank separator

Note: If the date and time separator field returns a blank, the separator may have been determined by the default for the specified value of the date and time format field.

Decimal position. The number of decimal positions. This entry is zero if the field is not numeric.

Default value. The default value for this field. The default value is defined by the DFT or DFTVAL keyword used in DDS, or by the WITH DEFAULT clause of the CREATE TABLE SQL statement. Some examples of returned data are:

SQL clause WITH DEFAULT value, where value is:	DDS keyword DFT(value), where value is:	Returned by API:
'ABC'	'ABC'	'ABC'
+999	+999	+999
	999	+999
999		999
-999	-999	-999
USER **Note:** This value means to use the User ID as the value.		USER
COCODE ('ABC')		COCODE ('ABC')

Digits. The number of digits. This entry is zero if the field is not numeric.

Displacement to all fields defined by FLDL0100 format. This field contains the offset from the beginning of this entry to the beginning of the data mapped by the FLDL0100 format.

Displacement to alternative field name. This field contains the offset from the beginning of this entry to the beginning of the alternative field name. This field is zero if there is no alternative field name.

Displacement to default value. This field contains the offset from the beginning of this entry to the beginning of the default data. This field is zero if there is no default data for the field.

Display or print column number. This field contains the column number specified in the DDS source or as calculated at compile-time by the DDS compiler. If this value was not calculated at compile-time, it will be set to -1.

Display or print row number. This field contains the row number specified in the DDS source. This value will be relative to the start of the format. If spacing keywords (such as SPACEA, SKIPA, and SLNO) were specified for the file, record or field, this value will be set to -1.

Edit code. The field edit code.

Edit word. The field edit word.

Edit word length. The length of the edit word used.

Field column headings CCSID.

0	There are no field column headings.

1-65,535	The CCSID for the field column headings.

Field data CCSID.

0	There is no field data.
1-65,535	The CCSID for the field data.

Field data encoding scheme. The encoding scheme associated with the field data CCSID.

Field edit words CCSID

0	There are no field edit words.
1-65,535	The CCSID for the field edit words.

Field length in bytes. The number of bytes the field occupies.

Field name. The name of the field the entry describes.

Field text description. The description of the field.

Field text description CCSID.

0	There is no field text description.
1-65,535	The CCSID for the field text description.

Record text description CCSID.

0	There is no record text description.
1-65,535	The CCSID for the record text description.

File library name specified. The library specified in the call to the API.

File library name used. The name of the library that contained the file.

File name specified. The file specified in the call to the API.

File name used. The name of the file where the member list was found.

File type. The type of file found.

BSCF	Binary synchronous communications (BSC) file
CMNF	Communications file
DDMF	Distributed data management file
DKTF	Diskette file
DSPF	Display file
ICFF	Intersystem communications function file
LF	Logical file
MXDF	Mixed file
PF	Physical file
PRTF	Printer file
SAVF	Save file
TAPF	Tape file

Format name. The content and format of the information returned for each field. The possible values are:

FLDL0100	Field information
FLDL0200	Field and default value information
FLDL0300	Field, alternative field name, and default value information

GENERATED BY. This value defines when DB2 will generate a value for the column when a row is inserted or updated in a table. If the identity column field is 0 and the ROWID column field is 0, this field is not applicable. The possible values are:

1	BY DEFAULT - Indicates that DB2 will generate a value for the column when a row is inserted or updated in a table unless a value is specified.
2	ALWAYS - Indicates that DB2 will always generate a value for the column when a row is inserted or updated in a table.

Graphic fields indicator. Whether this format contains graphic fields. The possible values are:

0	The format does not contain graphic fields.
1	The format does contain graphic fields.

Host variable indicator. Whether a query has been defined with a host variable or a parameter marker in place of a comparison operand (for example, FIELDA > :hostvar) or an arithmetic operand (for example, FIELDA * 10). Possible values follow:

0	The query definition does not contain a host variable or a parameter marker.
1	The query definition does contain a host variable or a parameter marker.

Identity column. This value specifies whether or not this column was created as an identity column. The possible values are:

0	This is not an identity column.
1	This is an identity column.

Identity column - CACHE. This value is the number of cached values. If the Identity column field is 0 (no value specified), this field is not applicable.

Identity column - CYCLE. This value specifies whether this identity column should continue to generate values after generating either its maximum or minimum value. If the identity column field is 0, this field is not applicable. The possible values are:

0	This identity column should not continue to generate values after generating either its minimum or maximum value.
1	This identity column should continue to generate values after generating either its minimum or maximum value.

Identity column - INCREMENT BY. This value specifies the interval between consecutive values of the identity column. This value applies to identity column fields. If this value is positive, this is an ascending identity column. If the value is negative, this is a descending identity column. If the identity column field is 0, this field is not applicable.

Identity column - MAXVALUE. This value specifies the maximum value at which an ascending identity column either cycles or stops generating values, or a descending identity column cycles to after reaching the minimum value. If the identity column field is 0, this field is not applicable.

Identity column - MINVALUE. This value specifies the minimum value at which a descending identity column either cycles or stops generating values, or an ascending identity column cycles to after reaching the maximum value. This value applies to identity column fields. If the identity column field is 0, this field is not applicable.

Identity column - ORDER. This value specifies whether the identity values must be generated in order of request. If the identity column field is 0, this field is not applicable. The possible values are:

0	Identity values need not be generated in order of request.
1	Identity values must be generated in order of request.

Identity column - Original START WITH. This value specifies the first value for the identity column as defined when the table was created. If the identity column field is 0, this field is not applicable.

Identity column - Current START WITH. This value specifies the first value for the identity column. If the START WITH value for the identity column was changed through the ALTER TABLE command, this value will show the current setting. If the identity column field is 0, this field is not applicable.

Input buffer position. The field's position within the input record.

Internal field name. The internal name used to identify the field the entry describes.

Length of alternative field name. The length of the alternative field name definition.

Length of default value. The length of the default value for this field. If the field has no default value, this field is zero.

Length of FLDL0200 format. The combined length of all data returned in format FLDL0200. Use this value to access the next list data entry.

Length of FLDL0300 format. The combined length of all data returned in format FLDL0300. Use this value to access the next list data entry.

Length of user-defined type name. The length of the user-defined type name. If the field has no user-defined type, this field is zero.

Maximum large object field length. The maximum length of data that can be contained for this field. This value applies to fields with the BLOB, CLOB or DBCLOB data type.

Null-capable fields indicator. Whether this format contains null-capable fields. The possible values are:

0	The format does not contain null-capable fields.
1	The format contains null-capable fields.

Null values allowed. Whether the result of this field can be the null value. The possible values are:

0	The field does not allow the null value.
1	The field does allow the null value.

Number of DBCS characters. The number of DBCS characters this field can contain if the field type is graphic data type. This value does not include the 2 bytes for the variable length portion of the field.

Offset to default value. The offset from the beginning of format FLDL0200 to the start of the default value for this field. If the field has no default value, this value is zero.

Output buffer position. The field's position within the output record.

Override processing. Whether overrides are to be processed. The possible values are:

0	No override processing
1	Override processing

Pad length for large object. This value applies fields with the BLOB, CLOB or DBCLOB data type. This value is the pad length of the buffer for this field.

Record format ID. The record format identifier.

Record format name specified. The record format specified in the call to the API.

Record format name used. The name of this record format.

Record length. The length of this record format.

Record text description. The text description of this record format.

Record text description CCSID.

0	There is no record text description.
1-65,535	The CCSID for the record text description.

Reserved. An ignored field.

ROWID column. This value specifies that this field has been designated as a ROWID column. The possible values are:

0	The ROWID attribute was not specified on this field.
1	This field was created with the ROWID attribute.

UCS-2 displayed field length. The display length of a field containing UCS-2 data. This value is zero if the field does not contain UCS-2 data. For information about UCS-2, see the Globalization topic in the iSeries Information Center.

User-defined type name. The name of the user-defined type object.

User-defined type library name. The library containing the user-defined type object.

Use. How the field is used:

I	Input
O	Output
B	Both input and output
N	Neither

Note: Use is from the program point of view and not necessarily the use specified in the DDS that created the file. For example, *DSPF subfile record fields return **B** even if the field is **O** in the DDS.

User space library name. The name of the library that contains the user space that is to receive the generated list.

User space name. The name of the user space that is to receive the generated list.

Variable length field indicator (overlay for MI mapping). Whether the field has been defined as *VARCHAR, VARLEN, or *VARGRF. Possible values are:

0	The field is not variable length.
1	The field is variable length.

Variable length fields in format indicator. Whether this format contains variable length fields. The possible values are:

0	The format does not contain variable length fields.
1	The format contains variable length fields.

Error Messages

Message ID	**Error Message Text**
CPF24B4 E	Severe error while addressing parameter list.
CPF3CF1 E	Error code parameter not valid.
CPF3C20 E	Error found by program &1.
CPF3C21 E	Format name &1 is not valid.
CPF3C22 E	Cannot get information about file &1.
CPF3C25 E	Value &1 for file override parameter is not valid.
CPF3C28 E	Record format &3 in file &1 not found.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C90 E	Literal value cannot be changed.
CPF8100 E	All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9800 E	All CPF98xx messages could be signaled. xx is from 01 to FF.

API introduced: V1R3

Top | “Database and File APIs,” on page 1 | APIs by category

List Open Files (QDMLOPNF)

Required Parameter Group:

1	Receiver variable
Output	Char(*)
2	Length of receiver variable
Input	Binary(4)
3	Format of receiver information
Input	Char(8)
4	Job identification information
Input	Char(*)
5	Format of job identification information
Input	Char(8)
6	Error code
I/O	Char(*)

Default Public Authority: *USE

Threadsafe: Yes

The List Open Files (QDMLOPNF) API generates a list of *FILE objects that are currently open in the job or that were opened by the thread that is specified in the job identification information input parameter.

Authorities and Locks

Job Authority
: This API must be called from within the job for which the information is being retrieved, or the caller of the API must be running under a user profile that is the same as the job user identity of the job for which the information is being retrieved. Otherwise, the caller of the API must be running under a user profile that has job control (*JOBCTL) special authority.

The **job user identity** is the name of the user profile by which a job is known to other jobs. It is described in more detail in the Work Management book on the V5R1 Supplemental Manuals Web site.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)

The receiver variable that is to receive the information requested. You can specify the size of the area to be smaller than the format requested as long as you specify the length of receiver variable parameter correctly. As a result, the API returns only the amount of data specified in the length of receiver variable.

Length of receiver variable
INPUT; BINARY(4)

The length of the receiver variable provided. The length of receiver variable parameter may be specified up to the size of the receiver variable specified in the user program. If the length of receiver variable parameter specified is larger than the allocated size of the receiver variable specified in the user program, the results are not predictable. The minimum length is 8 bytes.

Format of receiver information
INPUT; CHAR(8)

The format of the information returned in the receiver variable. The possible format name is:

OPNF0100	See "Format OPNF0100" for details on the list of files that this job or thread has open.

Job identification information
INPUT; CHAR(*)

The information that is used to identify the job or thread for which the list of open files is to be returned. See "Format JIDF0100" on page 34 for details.

Format of job identification information
INPUT; CHAR(8)

The format of the job identification information. The possible format name is:

JIDF0100	See "Format JIDF0100" on page 34 for details.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error code parameter.

Format OPNF0100

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Number of open files available
12	C	BINARY(4)	Offset to list of open files
16	10	BINARY(4)	Number of open files returned
20	14	BINARY(4)	Length of open file entry
24	18	CHAR(10)	Job name used
34	22	CHAR(10)	Job user name used
44	2C	CHAR(6)	Job number used
50	32	CHAR(8)	Thread identifier used
58	3A	CHAR(*)	Reserved

Offset			
Dec	**Hex**	**Type**	**Field**
These fields repeat, in the order listed, for the number of open files.		CHAR(10)	File name
		CHAR(10)	File library
		CHAR(10)	Member or device name
		CHAR(10)	File type
		CHAR(10)	Record format
		CHAR(10)	Activation group name
		CHAR(8)	Thread identifier
		CHAR(1)	Open option
		CHAR(3)	Reserved
		BINARY(8)	Activation group number
		BINARY(8)	Write count
		BINARY(8)	Read count
		BINARY(8)	Write/read count
		BINARY(8)	Other I/O count
		BINARY(8)	Relative record number
		BINARY(8)	Number of shared opens
		BINARY(4)	Object auxiliary storage pool number
		BINARY(4)	Library auxiliary storage pool number
		CHAR(10)	Object auxiliary storage pool name
		CHAR(10)	Library auxiliary storage pool name

Field Descriptions

Activation group name. The name of the activation group to which an open file is scoped. This field can contain the following special values:

**DFTACTGRP*	The file is scoped to the default activation group.
**JOB*	The file is scoped to the job, not a specific activation group.
**NEW*	The file is scoped to a *NEW activation group.

Activation group number. The number of the activation group to which an open file is scoped. This field will contain zero for files scoped to the job.

Bytes available. The number of bytes of data available to be returned. All available data is returned if enough space is provided.

Bytes returned. The number of bytes of data returned.

File library. The name of the library that contains the open file. If the file is an inline data file, blanks are returned. For DDM files, this is the library in which the DDM file is located.

File name. The name of the file that is open. This field will contain the value QINLINE for unnamed inline data files. For DDM files, this is the name of the DDM file.

File type. The type of file that is open.

BSCF	Binary Synchronous Communications (BSC) file
CMNF	Communications file
DDMF	Distributed Data Management file
DKTF	Diskette file (spooled and non-spooled)
DSPF	Display file
ICFF	Intersystem Communications Function file
LF	Logical file
MXDF	Mixed file
PF	Physical file
PRTF	Printer file (spooled and non-spooled)
SAVF	Save file
TAPF	Tape file
**INLINE*	Inline data file

Job name used. The name of the job for which open files were listed.

Job number used. The number of the job for which open files were listed.

Job user name used. The user name of the job for which open files were listed.

Length of open file entry. The length of each open file entry.

Library auxiliary storage pool name. The name of the auxiliary storage pool (ASP) in which the library of the open file resides. This field can contain the following special values:

**SYSBAS*	The library resides in the system ASP or a basic user ASP.
**N*	The ASP name could not be determined at this time.

Library auxiliary storage pool number. The number of the auxiliary storage pool (ASP) in which the library of the open file resides. Possible values are:

1	System ASP
2-32	Basic user ASPs
33-255	Independent ASPs

Member or device name. If the file type is physical (PF) or logical (LF), this is the name of the database member. If multiple member processing is being performed, the value *ALL is returned. For device files (BSCF, CMNF, DKTF, DSPF, ICFF, MXDF, PRTF, SAVF, or TAPF), this is the name of the last program device used for an I/O operation. This field is blank for device files when no I/O operation has been performed, and always for inline data files. If the file is a spooled file, the value *SPOOL is returned. If the file is a DDM file, blanks are returned.

Number of open files available. The number of open files available to be returned.

Number of open files returned. The number of complete open file entries that are returned.

Number of shared opens. The number of times the file was opened for shared processing. This field will contain zero for open operations that are not shared.

Object auxilliary storage pool name. The name of the auxilliary storage pool (ASP) in which the open file resides. This field can contain the following special values:

**SYSBAS*	The object resides in the system ASP or a basic user ASP.
**N*	The ASP name could not be determined at this time.

Object auxiliary storage pool number. The number of the auxiliary storage pool (ASP) in which the open file resides. Possible values are:

1	System ASP
2-32	Basic user ASPs
33-255	Independent ASPs

Offset to list of open files. The offset in bytes from the beginning of the receiver variable to the first open file entry.

Open option. The type of open operation that is performed:

0	The file was opened for input operations only.
1	The file was opened for output operations only.
2	The file was opened for all operations (input, output, update, and delete).

Other I/O count. Number of successful I/O operations of the following types:
- update
- delete
- change end-of-data
- force end-of-data
- force end-of-volume
- release record lock
- acquire or release program device

Read count. Number of successful read operations. If record blocking is not in effect for the file, this is the number of records. If record blocking is in effect for the file, this is the number of record blocks.

Record format. The name of the last record format that was used for an I/O operation to the file. If no record format name was used or no I/O operations have been performed, this field is blank.

Relative record number. Relative record number of the last record referred to by an I/O or open operation for database files. Zero is returned for nondatabase files and database files on which no I/O operations have been performed.

» **Reserved.** An ignored field.

«

Thread identifier. An 8-byte thread handle assigned by the system. It identifies the thread in which the file was opened.

Thread identifier used. The identifier of the thread for which open files were listed. A value of zero indicates open files were returned for all threads within the job.

Write count. The number of successful write operations. If record blocking is not in effect for the file, this is the number of records. If record blocking is in effect for the file, this is the number of record blocks.

Write/Read count. The number of successful write/read operations.

Format JIDF0100

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Job name
10	A	CHAR(10)	User name
20	14	CHAR(6)	Job number
26	1A	CHAR(16)	Internal job identifier
42	2A	CHAR(2)	Reserved
44	2C	BINARY(4)	Thread indicator
48	30	CHAR(8)	Thread identifier

Field Descriptions

Internal job identifier. The internal identifier for the job. The List Job (QUSLJOB) API returns this identifier. If you do not specify *INT for the job name parameter, this parameter must contain blanks. With this parameter, the system can locate the job more quickly than with the job name.

Job name. A specific job name or one of the following special values.

*	The job in which this program is running. The job number and user name must contain blanks.
**INT*	The internal job identifier locates the job. The job number and user name must contain blanks.

Job number. A specific job number, or blanks when the job name specified is a special value.

» **Reserved.** An unused field. This field must contain hexadecimal zeros. «

Thread identifier. The unique value used to identify the thread within the job. If the thread indicator is not 0, this field must contain hexadecimal zeroes.

Thread indicator. The value that is used to specify the thread within the job for which information is to be retrieved. The following values are supported:

0	The value in the thread identifier field should be used to locate the thread.
1	Information should be retrieved for the thread in which this program is running. The combination of the internal job identifier, job name, job number, and user name fields also must identify the job containing the current thread.
2	Information should be retrieved for the initial thread of the identified job.
3	Information should be retrieved for all threads within the specified job.

User name. A specific user profile name, or blanks when the job name specified is a special value.

Error Messages

Message ID	Error Message Text
CPF136A E	Job not active.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C21 E	Format name &1 is not valid.
CPF3C24 E	Length of the receiver variable is not valid.

Message ID	Error Message Text
CPF3C3B E	Value for parameter not valid.
CPF3C51 E	Internal job identifier not valid.
CPF3C52 E	Internal job identifier no longer valid.
CPF3C53 E	Job &3/&2/&1 not found.
CPF3C55 E	Job does not exist.
CPF3C57 E	Not authorized to retrieve job information.
CPF3C58 E	Job name specified is not valid.
CPF3C59 E	Internal identifier is not blanks and job name is not *INT.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9999 E	Function check.

API introduced: V5R1

Top | "Database and File APIs," on page 1 | APIs by category

List Record Formats (QUSLRCD) API

Required Parameter Group:

1	Qualified user space name
Input	Char(20)
2	Format name
Input	Char(8)
3	Qualified file name
Input	Char(20)
4	Override processing
Input	Char(1)

Optional Parameter Group:

5	Error code
I/O	Char(*)

Service Program Name: QUSLRCD

Default Public Authority: *USE

Threadsafe: No

The List Record Formats (QUSLRCD) API generates a list of record format information contained within the specified file and places the list in a specified user space. The created list replaces any existing information in the user space.

You can use the QUSLRCD API with database file types, such as *PF, *LF, and *DDMF, and device file types, such as *DSPF, *TAPF, *DKTF, *PRTF, *SAVF, and *ICFF.

Authorities and Locks

User Space Authority
*CHANGE

User Space Library Authority
*EXECUTE

File Library Authority
*USE

File Authority
*OBJOPR

User Space Lock
*EXCLRD

File Lock
*SHRRD

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)

The name of the user space that is to receive the generated list, and the library in which it is located. The first 10 characters contain the user space name, and the second 10 characters contain the library name. You can use these special values for the library name:

**CURLIB*	The job's current library
**LIBL*	The library list

Format name
INPUT; CHAR(8)

The format of the information returned. The possible format names are:

RCDL0100	Record format name only.
RCDL0200	Record format name and additional information. This format requires more system paging and takes longer to produce than the RCDL0100 format.
RCDL0300	Record format name and device file information. This format requires more system paging and takes longer to produce than the RCDL0100 format. This format is only applicable to device file types.

For more information, see "RCDL0100 List Data Section" on page 38, "RCDL0200 List Data Section" on page 38 or "RCDL0300 List Data Section" on page 38

Qualified file name
INPUT; CHAR(20)

The name of the file whose record format names are placed in the list, and the library in which it is located. The first 10 characters contain the file name, and the second 10 characters contain the library name. You can use these special values for the library name:

**CURLIB*	The job's current library
**LIBL*	The library list

Override processing
INPUT; CHAR(1)

Whether overrides are to be processed. The possible values are:

0	No override processing
1	Override processing

Optional Parameter

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Format of the Generated List

The record format list consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- A list data section

For details about the user area and generic header, see User Space Format for List APIs. For details about the other items, see the following sections. For descriptions of each field, see "Field Descriptions" on page 39.

When you retrieve list entry information from a user space, you must use the entry size returned in the generic header as a displacement to the next list entry. The size of each entry may be padded at the end. If you do not use the entry size, the result may not be valid. For examples of how to process lists, see API examples.

Input Parameter Section

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	File name specified
38	26	CHAR(10)	File library name specified
48	30	CHAR(1)	Override processing

Header Section

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	File name used
10	A	CHAR(10)	File library name used

Offset			
Dec	Hex	Type	Field
20	14	CHAR(10)	File type
30	1E	CHAR(50)	File text description
80	50	BINARY(4)	File text description CCSID
84	54	CHAR (13)	File creation date

RCDL0100 List Data Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Record format name

RCDL0200 List Data Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Record format name
10	A	CHAR(13)	Record format ID
23	17	CHAR(1)	Reserved
24	18	BINARY(4)	Record length
28	1C	BINARY(4)	Number of fields
32	20	CHAR(50)	Record text description
82	52	CHAR(2)	Reserved
84	54	BINARY(4)	Record text description CCSID

RCDL0300 List Data Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Record format name
10	A	CHAR(2)	Lowest response indicator
12	C	BINARY(4)	Buffer size
16	10	CHAR(20)	Record format type
36	24	CHAR(1)	Starting line number
37	25	CHAR(1)	Separate indicator area present

Field Descriptions

Buffer size. The user buffer size.

Record text description CCSID.

0	There is no record text description.
1-65,535	The CCSID for the record text description.

File creation date. The date of the file in the format CYYMMDDHHMMSS as follows:

C	Century, where 0 indicates years 19*xx* and 1 indicates years 20*xx*.
YY	Year
MM	Month
DD	Day
HH	Hour
MM	Minute
SS	Second

File library name specified. The name of the file library specified in the call to the API.

File library name used. The name of the library that contained the file. If the library requested was *LIBL or *CURLIB, this field contains the name of the library where the system found the file.

File name specified. The name of the file specified in the call to the API.

File name used. The name of the file whose record formats are listed. If override processing was requested, this is the actual file.

File text description. The text description of the file.

File text description CCSID.

0	There is no file text description.
1-65,535	The CCSID for the file text description.

File type. The type of file found:

BSCF	Binary synchronous communications (BSC) file
CMNF	Communications file
DSPF	Display file
DDMF	Distributed data management file
DKTF	Diskette file
ICFF	Intersystem communications function file
LF	Logical file
MXDF	Mixed file
PF	Physical file
PRTF	Printer file
SAVF	Save file
TAPF	Tape file

Lowest response indicator. The lowest response indicator in the file. The possible values are:

00	No response indicators in the file or response indicators are not applicable

01-99	Response indicator

Number of fields. The number of fields contained in this record format. You can use the List Field Description (QUSLFLD) API to retrieve field information about this record.

Override processing. Whether overrides are to be processed. The possible values are:

0	No override processing
1	Override processing

Record format name. The name of the format used to list records. The possible values are:

RCDL0100	Record format name only
RCDL0200	Record format name and additional information
RCDL0300	Record format name and device information

Record format name. The name of this record format.

Record length. The length of this record format.

Record text description. The text description of this record format.

Reserved. An ignored field.

Record format type. The type of this record format. The possible values are:

Normal	Normal record
SFL	Subfile record
SFLMSGRCD	Subfile message record
SFLCTL	Subfile control record
USRDFN	User-defined record
WINDOW	Window record

Separate indicator area present. The existence of a separate indicator area. The possible values are:

0	No indicator area
1	Indicator area

Starting line number. A starting line number was specified for this record format. The possible values are:

0	Starting line number is not specified.
1	Starting line number is specified.

User space library name. The name of the library that contains the user space that is to receive the generated list.

User space name. The name of the user space that is to receive the generated list.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF3C20 E	Error found by program &1.
CPF3C21 E	Format name &1 is not valid.
CPF3C22 E	Cannot get information about file &1.
CPF3C25 E	Value &1 for file override parameter is not valid.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C90 E	Literal value cannot be changed.
CPF8100 E	All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9800 E	All CPF98xx messages could be signaled. xx is from 01 to FF.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V1R3

Top | "Database and File APIs," on page 1 | APIs by category

Replay Database Operation (QDBRPLAY) API

Required Parameter Group:

1	Input template
Input	Char(*)
2	Length of input template
Input	Binary(4)
3	Input template format name
Input	Char(8)
4	Journal entry specific data
Input	Char(*)
5	Length of journal entry specific data
Input	Binary(4)
6	Rename exit program scratchpad
Input	Char(*)
7	Error code
I/O	Char(*)

Default Public Authority: *USE

Threadsafe: No

The Replay Database Operation (QDBRPLAY) API replays a database operation from a single journal entry.

Only database journal entries are supported. Since these journal entries can be quite large, the QjoRetrieveJournalEntries API should be used to retrieve the journal entry. If a journal entry is passed to QDBRPLAY that is not supported, the operation will fail. The following journal entries are supported:

Journal Code	Entry Type	Description
D	AC	Add Constraint
F	CB	Change Member
D	CG	Change File
D	CT	Create File
D	DC	Remove Constraint
F	DM	Remove Member
D	DT	Delete File
D	FM	Move File
D	FN	Rename File
D	GC	Change Constraint
D	GO	Change Owner
D	GT	Grant File
F	MC	Add Member
F	MN	Rename Member
F	RM	Reorganize Member
D	RV	Revoke File
D	TC	Add Trigger
D	TD	Remove Trigger
D	TG	Change Trigger
D	TQ	Refresh Table

You can use the QDBRPLAY API with database objects only. DDM files are not supported. File overrides do not affect the specified object names. The API does not run under commitment control even if the original journal entry was performed as part of a commitable transaction. If the specified file does not have the same File Level Identifier or Member Level Identifier as the file for which the journal entry was originally written, a warning will sent to the job log and the operation will continue.

Authorities and Locks

Object Library Authority

*EXECUTE

Note: Additionally, the same authority is necessary as the original operation. For example, if the journal entry is Create File, *ADD is required.

Object Authorities

The same authority is necessary as was required for the original operation. For example, if the journal entry is Delete File, *OBJEXIST is required.

Object Lock

*EXCL or *EXCLRD for *FILE objects.

Note: The same locks are necessary as were required during the original operation. For example, if the journal entry is Delete File, *EXCL is required. If the journal entry is Remove Member, *EXCLRD is required. Because an exclusive lock is required, concurrent applications may not access the file object identified by this API till the API has ended and any concurrent applications that hold a conflicting lock cause the API to fail.

Rename Exit Program Library Authority

*EXECUTE

Rename Exit Program Authority

*EXECUTE

Required Parameter Group

Input template
INPUT;CHAR(*)

A structure that contains the input options used to replay the operation from the journal entry. For the format of this parameter, see "DBRR0100 Format" on page 44.

Length of input template
INPUT; BINARY(4)

A variable that contains the length of the input template. The length must be greater than zero and large enough to contain all the template fields up to and including Disable Triggers. The length must not be larger than 32767.

Input template format name
INPUT; CHAR(8)

The format of the input template being used. The possible value is:

DBRR0100	Basic template

For more information, see "DBRR0100 Format" on page 44.

Journal entry specific data
INPUT;CHAR(*)

The entry specific data from a database journal entry. If the original journal entry contained any additional journal entry specific data that was addressed by a pointer, that data must be moved so that the entry specific data includes all the entry specific data immediately preceding the pointer concatenated with the entry specific data that the pointer addresses. For example, if a teraspace pointer is returned on the QjoRetrieveJournalEntries API immediately after Entry Specific Data A and points to additional Entry Specific Data B:

Entry Specific Data A
Teraspace pointer to additional data B

This must be passed to the API as:

Entry Specific Data A
Additional Entry Specific Data B

See the Journal management topic in the iSeries Information Center for information on Entry Specific Data and Additional Entry Specific Data.

Note: The entry specific data for the database operations may be quite large, but is never larger than what will fit in a 16 megabyte space.

Length of journal entry specific data

INPUT;BINARY(4)

The length of the entry specific data from a database journal entry. The length must be greater than zero and must be the length of the actual entry specific data provided when the journal entry was originally written (including any additional journal entry specific data). The length must not be larger than 16 773 120.

Rename exit program scratchpad

INPUT; CHAR(*)

An area which is passed to the rename exit program. The area can contain any information that the caller of the API wishes to pass on to the rename exit program. A rename exit program scratchpad must be passed even if the rename exit program is not specified.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

DBRR0100 Format

The following table shows the format of the input template parameter for the DBRR0100 format. For detailed descriptions of the fields in the table, see “Field Descriptions.”

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(1)	Journal code
1	1	CHAR(2)	Entry type
3	3	CHAR(10)	Rename exit program name
13	D	CHAR(10)	Rename exit program library name
23	17	CHAR(1)	Disable triggers
24	18	CHAR(*)	Reserved

Field Descriptions

Disable triggers. The disable trigger indicator controls whether new triggers that are added as a result of replaying a TC journal entry should be automatically disabled.

0	Do not disable new triggers.
1	Disable new triggers.

Entry type. The journal entry type from the journal entry of the operation to replay. See the Journal management topic for more information on Entry type.

Note that a journal entry with an entry type of MN is written to the journal for both a rename member operation and for the internal operations performed as part of a rename file. If a journal entry type of MN that was written to the journal as part of a rename file operation is passed to this API, it will be ignored and no error will be returned.

Journal code. The journal code from the journal entry of the operation to replay. See the Journal management topic for more information on journal codes.

D Database file operation.

F Database file member operation.

Rename exit program name. The name of the rename exit program.

**NONE*
A rename exit program is not provided. The names of any objects referenced during the replay of the operation will be the same as the object names that were referenced when the journal entry was originally written to the journal.

program-name
The name of the program to call which may provide a different name for any object referenced in the journal entry. When a rename exit program is specified, each name referenced during the replay of the operation will be passed to the rename exit program. The names passed to the rename exit program may be short names or long SQL names. The same name may be passed to the exit program more than once if it is referenced in the internal journal entry specific data more than once. If the names are changed by the rename exit program, the names are case sensitive and must conform to any OS/400 and SQL rules for object names. For example, if the new name of the object should be ″a″, it must be returned from the rename exit program with the quote delimiters and a lower case a. If the new name of the object should be A, it must be returned from the rename exit program in upper case without redundant quote delimiters. The following restrictions apply to referenced objects:

- The library name of an SQL type or an SQL function can be changed. The SQL type name or SQL function name cannot be changed.
- Any references to objects in the body of a program object or trigger are not renamed.
- Any references to objects in the body of an SQL view other than the based on files themselves are not renamed.
- The name of a data dictionary must be the same name as its containing library.
- The library name of a constraint must be the same as the library name of its file.

Two parameters are passed to the rename exit program. The first parameter is the Rename Exit Program Parameter Template. The second parameter is the Rename Exit Program Scratchpad. For more information, see "**Rename Exit Program Parameter**" on page 46. The exit program may inspect the name passed and leave the name as is, or it may change the name in the Rename Exit Program Parameter to an alternate name.

For example, the journal entry contains the create of an SQL table that contains a primary key constraint. The exit program will be called twice. The first call will pass the table name and library. The second call will pass the constraint name and library. If the name is a reference to another object and is changed to reference an object that does not exist, the replay of the operation will fail.

If the rename exit program returns an exception, the replay operation will fail.

Rename exit program library name. The library that contains the rename exit program, if any. The rename exit program library name must be blank if the value of the rename exit program name is *NONE.

library-name
The library name of the rename exit program.

Reserved. A reserved field. It must contain hexadecimal zeroes.

Rename Exit Program Parameter

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Length of rename exit program parameter
4	4	BINARY(4)	Length of object name
8	8	BINARY(4)	Length of object library
12	C	BINARY(4)	Object type
16	10	CHAR(258)	Object name
274	112	CHAR(258)	Object library name
538	21A	CHAR(*)	Reserved

Field Descriptions

Length of rename exit program parameter. The length of the structure passed to the rename exit program. The rename exit program must not modify this value.

Length of object library name. The length of the library name of the referenced object. If the rename exit program modifies this length, it must be greater than zero and not greater than 10. It must reflect the number of characters in the new object library name.

Length of object name. The length of the name of the referenced object. If the rename exit program modifies this length, it must be greater than zero and must reflect the number of bytes in the new object name.

If the value that was passed to the rename exit program is less than or equal to 10 and the rename exit program modifies this length, the new length must also be less than or equal to 10 (a short system name cannot be renamed to a long SQL name). If the value that was passed to the rename exit program is greater than 10 the rename exit program must not modify this length (a long SQL name cannot be renamed to a short name or a long SQL name of a different length).

Object library name. The library name of the referenced object. If the rename exit program modifies the library name, it must be a valid library name.

library-name
: The library name of the referenced object.

Object name. The name of the referenced object. If the rename exit program modifies the object name, it must be a valid short object name or a valid long SQL object name.

If the name passed to the exit program is not a long SQL object name, the rename exit program must not rename the object to a long SQL object name. If the name passed to the exit program is a long SQL object name, the rename exit program may rename the long SQL name to another long SQL name, but the length of the new long SQL name must be the same as the long SQL name passed to the rename exit program.

object-name
: The name of the referenced object. The name may be either a short (10 character) object name or a long (258 character) SQL object name.

Object type. The type of the referenced object. The following values may be passed to the exit program for the object type:

1	The object attribute is a constraint.

2	The object is an SQL function (*PGM or *SRVPGM).
3	The object is a file (*FILE).
4	The object is a program object (*PGM).
5	The object attribute is a trigger.
6	The object is an SQL type (*SQLUDT).
7	The object is an data dictionary (*DTADCT).
8	The object is a sort sequence or translate table (*TBL).
9	The object is a node group (*NODGRP).
10	The object is a journal (*JRN).

The rename exit program must not modify this value.

While the rename exit program will be called for referenced journals, data dictionaries, and SQL types, these objects themselves cannot cannot be renamed. Furthermore, a library that contains one of these object types cannot be renamed.

Reserved. A reserved field. The rename exit program must not modify this value.

Usage Notes

If a file is created as a result of replaying a Create File (CT) entry:

- If journaling was implicitly started when an SQL table was originally created, the replay operation will also implicitly start journaling to the same journal.
- The File level identifier for the created file will be the same as the File level identifier of the file when it was originally created.
- Any public authority that was specified by the AUT keyword when a file is created via a CL command will be granted when the file is created.

If a member is added as a result of replaying an Add Member operation (MC), the Member level identifier for the added member will be the same as the Member level identifier of the member when it was originally added.

Error Messages

CPF24B4 E	Severe error while addressing parameter list.
CPF3C21 E	Format name &1 is not valid.
CPF3C39 E	Value for reserved field not valid.
CPF3C3A E	Value for parameter &2 for API &1 not valid.
CPF3C3B E	Value for parameter &2 for API &1 not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF3200 E	All CPF32xx messages could be returned. xx is from 01 to FF.
CPF8100 E	All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9800 E	All CPF98xx messages could be signaled. xx is from 01 to FF.
SQL0113 E	Name is not valid.

« API introduced: V5R3

Top | "Database and File APIs," on page 1 | APIs by category

Retrieve Database File Description (QDBRTVFD) API

Required Parameter Group:

1	Receiver variable	
Output	Char(*)	
2	Length of receiver variable	
Input	Binary(4)	
3	Qualified returned file name	
Output	Char(20)	
4	Format name	
Input	Char(8)	
5	Qualified file name	
Input	Char(20)	
6	Record format name	
Input	Char(10)	
7	Override processing	
Input	Char(1)	
8	System	
Input	Char(10)	
9	Format type	
Input	Char(10)	
10	Error Code	
I/O	Char(*)	

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 133.

The Retrieve Database File Description (QDBRTVFD) API allows you to get complete and specific information about a file on a local or remote system. The information is returned to a receiver variable in either a file definition template or a format definition mapping. The file definition template provides more complete information about a database file than the Display File Description (DSPFD) command. The format definition provides complete information on the record formats of the file.

The format definition is used with the Query (QQQQRY) API to get data from a file. You can run the QDBRTVFD API to build a format definition that is later used to run a query. This format definition can be used several times to extract information from a database, making the Query API run faster. If the format definition is not created prior to running a query, the QQQQRY API must create one when it runs.

Authorities and Locks

Library Authority
    *EXECUTE

File Authority
    *OBJOPR

File Lock
*SHRNUP

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)

The receiver variable that is to receive the information requested. You can specify the size of the area smaller than the format requested as long as you specify the length of receiver variable parameter correctly. As a result, the API returns only the data the area can hold.

Length of receiver variable
INPUT; BINARY(4)

The length of the receiver variable provided. The length of receiver variable parameter may be specified up to the size of the receiver variable specified in the user program. If the length of receiver variable parameter specified is larger than the allocated size of the receiver variable specified in the user program, the results are not predictable. The minimum length is 8 bytes.

Qualified returned file name
OUTPUT; CHAR(20)

The actual qualified file name from which the file description has been extracted. If an override is active this file and library name may be different from the one entered with the API.

Format name
INPUT; CHAR(8)

The content and format of the information to be returned about the specified file, member, or format. You can use the following format names:

FILD0100	File definition template
FILD0200	Format definition template
FILD0300	Key field information template
FILD0400	Trigger information template

See "Format of Generated Information" on page 53 for a description of these formats.

Qualified file name
INPUT; CHAR(20)

The name of the file about which the information is to be extracted and the library in which it is located. The first 10 characters contain the file name, and the second 10 characters contain the library name.

You can use the following special values for the library name:

**CURLIB*	The job's current library
**LIBL*	The library list

Record format name
INPUT; CHAR(10)

The name of the record format in the specified file that is to be used to generate the file description. (This parameter is used only with format FILD0200.)

You can use the following special value for the record format name

**FIRST*	The first record format found

Override processing

INPUT; CHAR(1)

Whether overrides are to be processed. The following values are used:

0	No override processing
1	Override processing

System

INPUT; CHAR(10)

Whether the information that is returned is about a file on either a local or remote system, or both. The possible values are:

**LCL*	The information returned is about local files only.
**RMT*	The information returned is about remote files only.
**FILETYPE*	The information returned is about files on both the local and remote systems. For DDM files, the information returned is about the remote file that was named on the RMTFILE parameter of the Create DDM File (CRTDDMF) command.

Format type

INPUT; CHAR(10)

Whether the logical formats returned are internal or external. (This parameter is used only with format FILD0200.) A description and examples of the internal (*INT) and external (*EXT) formats follow:

**EXT*	The formats returned are external. If the specified file is a logical file, the format returns data for the logical fields defined in the logical record format. If the specified file is a physical file, the internal and external field names are the same.
**INT*	The formats returned are internal. If the specified file is a logical file, the format returns data for the fields on which the logical fields are based. If the specified file is a physical file, the internal and external field names are the same.

The following are DDS, *EXT, and *INT format type examples: For a logical file definition of **(1)** that is based on a physical file definition of **(2)**, a format type of *EXT would return **(3)** and a format type of *INT would return **(4)**.

Format Type Example DDS

Logical file definition **(1)**:

R CONCAT1

PFILE(PF1)

LFLD1

RENAME(FLD1)

FLD2

CATFLD

CONCAT(FLD1 FLD2 FLD3)

K CATFLD

Physical file definition **(2)**:

FLD1

5A

FLD2

10A

FLD3

5A

K FLD1

Format Type *EXT Example
(3)

Record format name
CONCAT1

Record length
35

Number of fields
3

Internal field name 1
FLD1

External field name 1
LFLD1

Length of field 1
5

Internal field name 2
FLD2

External field name 2
FLD2

Length of field 2
10

Internal field name 3
FLD1

External field name 3
CATFLD

Length of field 3
20

Format Type *INT Example
(4)

Record format name
CONCAT1

Record length
35

Number of fields
5

Internal field name 1
FLD1

External field name 1
LFLD1

Length of field 1
5

Internal field name 2
FLD2

External field name 2
FLD2

Length of field 2
10

Internal field name 3
FLD1

External field name 3
CATFLD

Length of field 3
5

Internal field name 4
FLD2

External field name 4
CATFLD

Length of field 4
10

Internal field name 5
FLD3

External field name 5
CATFLD

Length of field 5
5

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Format of Generated Information

The QDBRTVFD API can be used to provide information in the following formats:

FILD0100	File definition template
FILD0200	Format definition template
FILD0300	Key field information template
FILD0400	Trigger information template

The following sections provide an overview of each of these formats. If an offset equals zero in the returned information, there is no corresponding structure associated with it.

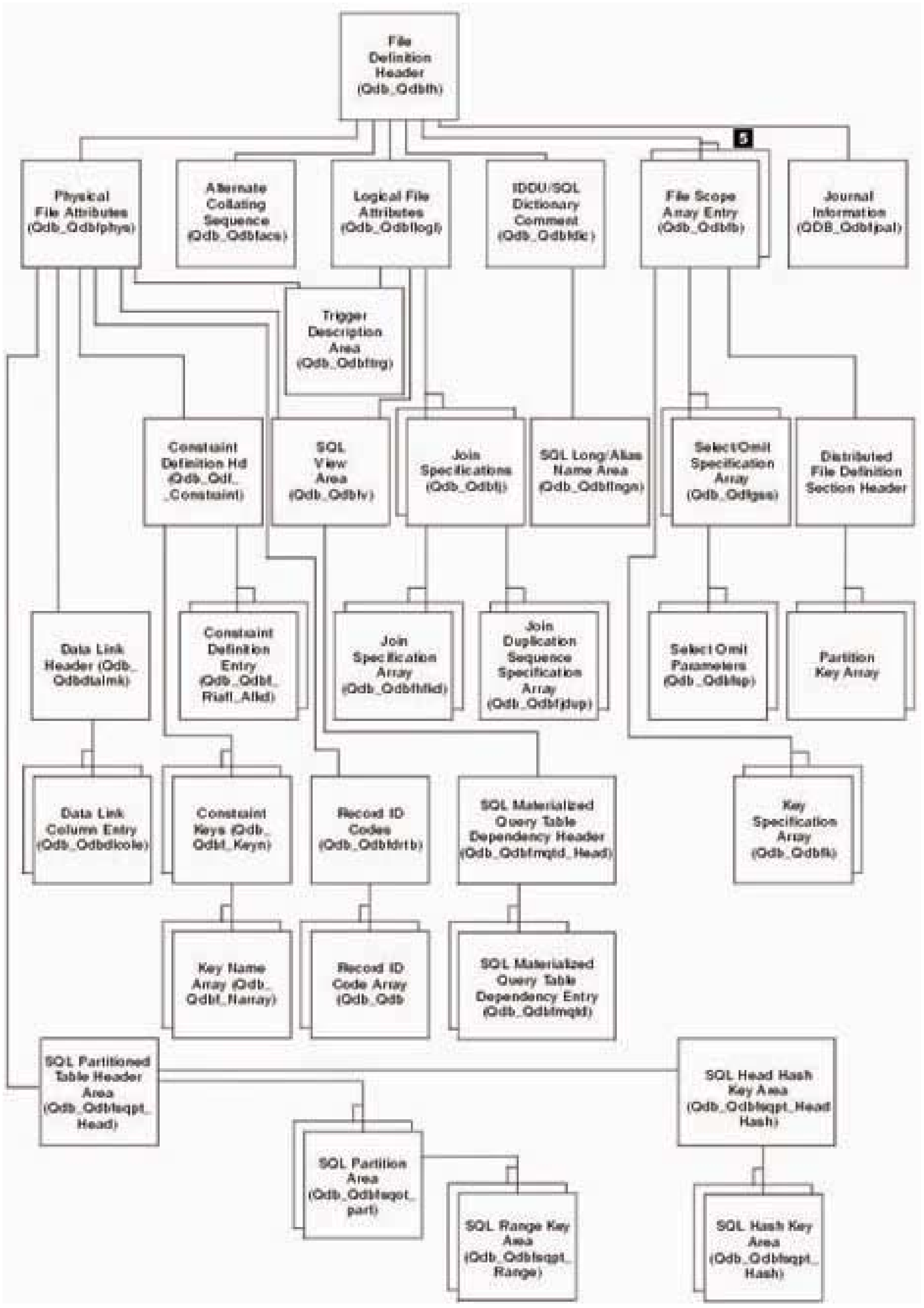
The asterisk (*) in the *Field* column represents a reserved field. No variable is associated with these reserved fields.

FILD0100 Format (File Definition Template (FDT) header)

FILD0100 provides detailed information about how the file is built. FILD0100 Format (page 54) shows how this information is organized. When more than one entry can appear, the figure indicates this as in **(5)**.

Descriptions of the fields in this structure follow FILD0100 Format (page 54). The include source is supplied on the system, in source file H, member name QDBRTVFD, in the QSYSINC library. The field names in the following tables apply only to the ILE C include. Refer to Include files and the QSYSINC Library for the names of the OPM and ILE RPG and COBOL includes.

FILD0100 Format


File Definition Header (Qdb_Qdbfh)
5
Physical File Attributes (Qdb_Qdbfphys)
Alternate Collating Sequence (Qdb_Qdbfacs)
Logical File Attributes (Qdb_Qdbflogl)
IDDU/SQL Dictionary Comment (Qdb_Qdbfdic)
File Scope Array Entry (Qdb_Qdbfb)
Journal Information (QDB_Qdbfjoal)
Trigger Description Area (Qdb_Qdbftrg)
Constraint Definition Hd (Qdb_Qdf_Constraint)
SQL View Area (Qdb_Qdbfv)
Join Specifications (Qdb_Qdbfj)
SQL Long/Alias Name Area (Qdb_Qdbflngn)
Select/Omit Specification Array (Qdb_Qdfgss)
Distributed File Definition Section Header
Data Link Header (Qdb_Qdbdtalnk)
Constraint Definition Entry (Qdb_Qdbf_Rtall_Alkd)
Join Specification Array (Qdb_Qdbfjfld)
Join Duplication Sequence Specification Array (Qdb_Qdbfjdup)
Select Omit Parameters (Qdb_Qdbfsp)
Partition Key Array
Data Link Column Entry (Qdb_Qdbdlcole)
Constraint Keys (Qdb_Qdbf_Keyn)
Record ID Codes (Qdb_Qdbfdrtb)
SQL Materialized Query Table Dependency Header (Qdb_Qdbfmqtd_Head)
Key Specification Array (Qdb_Qdbfk)
Key Name Array (Qdb_Qdbf_Narray)
Record ID Code Array (Qdb_Qdb
SQL Materialized Query Table Dependency Entry (Qdb_Qdbfmqtd)
SQL Partitioned Table Header Area (Qdb_Qdbfsqpt_Head)
SQL Head Hash Key Area (Qdb_Qdbfsqpt_Head Hash)
SQL Partition Area (Qdb_Qdbfsqpt_part)
SQL Range Key Area (Qdb_Qdbfsqpt_Range)
SQL Hash Key Area (Qdb_Qdbfsqpt_Hash)

«

File Definition Header (Qdb_Qdbfh)

Qdb_Qdbfh is the first structure and is located at offset zero of the returned data.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		BINARY(4)	Qdbfyret	Length of the data returned in bytes.
4	4		BINARY(4)	Qdbfyavl	Number of bytes provided for the file definition data.
8	8		BIT(16)	Qdbfhflg	Attributes bytes.
8	8	0	BIT(2)	Reserved_1	Reserved.
8	8	2	BIT(1)	Qdbfhfpl	Type of file. If on, the file is a logical database file. If off, a physical database file.
8	8	3	BIT(1)	Reserved_2	Reserved.
8	8	4	BIT(1)	Qdbfhfsu	File type (FILETYPE). If on, the file is a source file (*SRC). If off, a data file (*DATA).
8	8	5	BIT(1)	Reserved_3	Reserved.
8	8	6	BIT(1)	Qdbfhfky	Access path. If on, the file has a keyed sequence access path. If off, an arrival sequence access path.
8	8	7	BIT(1)	Reserved_4	Reserved.
9	9	0	BIT(1)	Qdbfhflc	Record format level check (LVLCHK). If on, the record format level identifiers are checked when the file is opened (*YES) if off, they are not checked when the file is opened (*NO).
9	9	1	BIT(1)	Qdbfkfso	Select/omit. If on, the file is a select/omit logical file.
9	9	2	BIT(4)	Reserved_5	Reserved.
9	9	6	BIT(1)	Qdbfigcd	Double-byte character set (DBCS) or Graphic data. If on, the file's record format(s) contains DBCS or Graphic data fields.
9	9	7	BIT(1)	Qdbfigcl	Double-byte character set (DBCS) or Graphic literals. If on, the file's record format(s) contains DBCS or Graphic literals.
10	A		CHAR(4)	Reserved_7	Reserved.
14	E		BINARY(2)	Qdbflbnum	Number of data members. 0 indicates an externally described physical file or a program described physical file that is not linked to a data dictionary. 1 through 32 indicates the number of data dictionary record formats for a program described physical file that is linked to a data dictionary or the number of based-on physical records for a logical file.
16	10		CHAR(13)	Qdbfkdat	Keyed sequence access path description. If this file has an arrival sequence access path, these fields are not applicable.
16	10		BINARY(2)	Qdbfknum	Number of key fields for the file. 1 through 120.
18	12		BINARY(2)	Qdbfkmxl	Maximum key length for the file. 1 through 2000.
20	14		CHAR(1)	Qdbfkflg	Keyed sequence access path attributes.
20	14	0	BIT(1)	Reserved_8	Reserved
20	14	1	BIT(1)	Qdbfkfcs	Alternate collating sequence (ALTSEQ). If on, an alternate collating sequence table is specified for the file.
20	14	2	BIT(4)	Reserved_9	Reserved.
20	14	6	BIT(1)	Qdbfkfrc	Force keyed access path (FRCACCPTH). If on, the access path and changed records are forced to auxiliary storage when the access path is changed (*YES).

Offset		Bit	Type	Field	Description
Dec	Hex				
20	14	7	BIT(1)	Qdbfkflt	Floating point key indicator. If on, the access path for the file contains floating point keys.
21	15		CHAR(1)	Qdbfkfdm	Access path maintenance (MAINT). *I* Immediate maintenance (*IMMED) *D* Delayed maintenance (*DLY) *R* Rebuild maintenance (*REBLD)
22	16		CHAR(8)	Reserved_10	Reserved.
30	1E		CHAR(10)	Qdbfhaut	Public authority (AUT). **CHANGE* Public change authority **ALL* Public all authority **USE* Public use authority **EXCLUDE* Public exclude authority *authorization-list-name* The name of the authorization list whose authority is used for the file. This is the original public authority that the file was created with, not the current public authority for the file.
40	28		CHAR(1)	Qdbfhupl	Preferred storage unit (UNIT). *X'00'* The storage space and its members can be allocated on the available auxiliary storage unit (*ANY). *X'01' through X'FF'* The unit identifier of an auxiliary storage unit on the system.
41	29		BINARY(2)	Qdbfhmxm	Maximum members (MAXMBRS). *0* No maximum is specified; 32,767 is used (*NOMAX). *1 through 32,767* The maximum number of members the file can have.
43	2B		BINARY(2)	Qdbfwtfi	Maximum file wait time (WAITFILE). *-1* The default wait time specified in the class description is used (*CLS). *0* The program does not wait for the file; an immediate allocation is required (*IMMED). *1 through 32,767* The number of seconds a program waits for the file.
45	2D		BINARY(2)	Qdbfhfrt	Records to force a write (FRCRATIO). *0* There is force write ratio. *1 through 32,767* The number of inserted, updated, or deleted records that are explicitly forced to storage.
47	2F		BINARY(2)	Qdbfhmnum	Number of members, 0 through 32,767.
49	31		CHAR(9)	Reserved_11	Reserved.

Offset					
Dec	Hex	Bit	Type	Field	Description
58	3A		BINARY(2)	Qdbfbrwt	Maximum record wait time (WAITRCD). *-2* The default wait time allowed by the system is used (*NOMAX). *-1* The program does not wait for the record, an immediate allocation is required (*IMMED). *1 through 32,767* The number of seconds a program waits for the record.
60	3C		CHAR(1)	Qaaf	Additional attribute flags.
60	3C	0	BIT(7)	Reserved_12	Reserved
60	3C	7	BIT(1)	Qdbfpgmd	Program described file indicator. If on, the file is program described.
61	3D		BINARY(2)	Qdbffmtnum	Total number of record formats, 1 through 32.
63	3F		CHAR(2)	Qdbfhfl2	Additional attribute flags
63	3F	0	BIT(1)	Qdbfjnap	Access path journaled.
63	3F	1	BIT(1)	Reserved_13	Reserved.
63	3F	2	BIT(4)		File capability/operation flags.
63	3F	2	BIT(1)	Qdbfrdcp	Allow read operation. If on, records are not allowed to be read from the file.
63	3F	3	BIT(1)	Qdbfwtcp	Allow write operation. If on, records are not allowed to be written to the file.
63	3F	4	BIT(1)	Qdbfupcp	Allow update operation (ALWUPD). If on, records are not allowed to be updated in the file (*NO).
63	3F	5	BIT(1)	Qdbfdlcp	Allow delete operation (ALWDLT). If on, records are not allowed to be deleted from the file (*NO).
63	3F	6	BIT(9)	Reserved_14	Reserved.
64	40	7	BIT(1)	Qdbfkfnd	Null values cause duplicates indicator (UNIQUE). Only valid if Qdbfpact is equal to 'KU'. If on, null values do not cause duplicate keys in the file access path(s) (*EXCNULL).

Offset					
Dec	Hex	Bit	Type	Field	Description
65	41		BINARY(2)	Qdbfvrm	First supported version release modification level. New database support is used in the file that will prevent it from being saved and restored to a prior version, release, and modification level. *X'0000'* Pre-Version 2 Release 1 Modification 0 file. *X'1500'* Version 2 Release 1 Modification 0 V2R1M0 file. *X'1501'* Version 2 Release 1 Modification 1 V2R1M1 file. *X'1600'* Version 2 Release 2 Modification 0 V2R2M0 file. *X'1700'* Version 2 Release 3 Modification 0 V2R3M0 file. *X'1F00'* Version 3 Release 1 Modification 0 V3R1M0 file. *X'2000'* Version 3 Release 2 Modification 0 V3R2M0 file. *X'2400'* Version 3 Release 6 Modification 0 V3R6M0 file. *X'2500'* Version 3 Release 7 Modification 0 V3R7M0 file. *X'2900'* Version 4 Release 1 Modification 0 V4R1M0 file. *X'2A00'* Version 4 Release 2 Modification 0 V4R2M0 file. *X'2B00'* Version 4 Release 3 Modification 0 V4R3M0 file. *X'2C00'* Version 4 Release 4 Modification 0 V4R4M0 file.
67	43		CHAR(2)	Qaaf2	Additional attribute flags.
67	43	0	BIT(1)	Qdbfhmcs	Multiple coded character set identifier indicator (CCSID). If on, the file has more than one CCSID for its input and output character type fields. If the file has no character type fields, this bit is off.
67	43	1	BIT(1)	Reserved_15	Reserved.
67	43	2	BIT(1)	Qdbfknll	Allow null value key indicator (ALWNULL). If on, null value keys are allowed.
67	43	3	BIT(1)	Qdbf_nfld	Allow null value data (ALWNULL). If on, the file record format(s) allow null value fields.
67	43	4	BIT(1)	Qdbfvfld	Variable length data (VARLEN). If on, the file record format(s) contain variable length fields.
67	43	5	BIT(1)	Qdbftfld	Date/time/timestamp data. If on, the file record format(s) contain date, time, or timestamp fields.
67	43	6	BIT(1)	Qdbfgrph	Graphic data. If on, the file record formats contain graphic fields.
67	43	7	BIT(1)	Qdbfpkey	Primary key (*PRIKEY). If on, the access path for the file is a primary key.
68	44	0	BIT(1)	Qdbfunqc	Unique constraint (*UNQCST). If on, the access path for the file is a unique constraint.
68	44	1	BIT(2)	Reserved_118	Reserved.

Offset					
Dec	Hex	Bit	Type	Field	Description
68	44	3	BIT(1)	Qdbfapsz	Access path size (ACCPTHSIZ). If on (*MAX1TB), all access paths associated with this file will be allowed to occupy a maximum of 1 terabyte (1 099 511 627 776 bytes) of auxiliary storage. If off (*MAX4GB), all access paths associated with this file will be allowed to occupy a maximum of 4 gigabytes (4 294 966 272 bytes) of auxiliary storage.
68	44	4	BIT(1)	Qdbfdisf	Distributed file. If on, the file is a distributed file.
68	44	5	BIT(1)	Reserved_68	Reserved.
68	44	6	BIT(1)	Reserved_69	Reserved.
68	44	7	BIT(1)	Reserved_70	Reserved.
69	45		CHAR(13)	Qdbfhcrt	File level identifier. The date of the file in internal standard format (ISF), CYYMMDDHHMMSS.
82	52		CHAR(52)	Qdbfhtx	File text description.
82	52		CHAR(2)	Reserved_18	Reserved.
84	54		CHAR(50)	Qdbfhtxt	Text description (TEXT)
134	86		CHAR(13)	Reserved_19	Reserved.
147	93		CHAR(30)	Qdbfsrc	Source file fields. Must be hexadecimal zeros if there is no source file information.
147	93		CHAR(10)	Qdbfsrcf	Source file name.
157	9D		CHAR(10)	Qdbfsrcm	Source file member name.
167	A7		CHAR(10)	Qdbfsrcl	Source file library name.
177	B1		CHAR(1)	Qdbfkrcv	Access path recovery (RECOVER). *A* The file access path is built after the IPL is completed (*AFTIPL). *N* The file access path is built when the file is next opened (*NO). *S* The file access path is built during the IPL (*IPL).
178	B2		CHAR(23)	Reserved_20	Reserved.
201	C9		BINARY(2)	Qdbftcid	Coded character set identifier (CCSID) for text description. *0* There is no file text description. *1 through 65,535* The file text description CCSID.
203	CB		CHAR(2)	Qdbfasp	Auxiliary storage pool (ASP). *X'0000'* The file is located on the system auxiliary storage pool. *X'0002' through X'0010'* On which user auxiliary storage pool the file resides.
205	CD		CHAR(1)	Qdbfnbit	Complex objects flags.
205	CD	0	BIT(1)	Qdbfhudt	If on, the file record format has a user-defined type field.
205	CD	1	BIT(1)	Qdbfhlob	If on, the file record format has a large object field.
205	CD	2	BIT(1)	Qdbfhdtl	If on, the file record format has a datalink field. A **datalink** is a field data type that is used to point to another object that contains the data for that field.
205	CD	3	BIT(1)	Qdbfhudf	If on, the file uses a user-defined function.

Offset					
Dec	Hex	Bit	Type	Field	Description
205	CD	4	BIT(1)	Qdbfhlon	If on, the file has a datalink field with FILE LINK CONTROL.
205	CD	5	BIT(1)	Qdbfhlop	If on, the file is a logical file without any large object fields, but the based-on physical file has a large object field.
205	CD	6	BIT(1)	Qdbfhdll	If on, the file is a logical file without any datalink fields, but the based-on physical file has a datalink field.
205	CD	7	BIT(1)	Reserved_21	Reserved.
206	CE		BINARY(2)	Qdbfmxfnum	Maximum number of fields, 1 through 8000. Indicates the number of fields in the file's record format that contains the largest number of fields.
208	D0		CHAR(76)	Reserved_22	Reserved.
284	11C		BINARY(4)	Qdbfodic	Offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the IDDU/SQL Data Dictionary area, Qdbfdic (page 78).
288	120		CHAR(14)	Reserved_23	Reserved.
302	12E		BINARY(2)	Qdbffigl	File generic key length, 0 through 2000. The length of the key before the first *NONE key field for the file. If this file has an arrival sequence access path, this field is not applicable.
304	130		BINARY(2)	Qdbfmxrl	Maximum record length, 1 through 32,766. The length of the record in the file's record format that contains the largest number of bytes.
306	132		CHAR(8)	Reserved_24	Reserved.
314	13A		BINARY(2)	Qdbfgkct	File generic key field count, 0 through 120. The count of the number of key fields before the first *NONE key field for the file. If this file has an arrival sequence access path, this field is not applicable.
316	13C		BINARY(4)	Qdbfos	Offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the file scope array, Qdbfb (page 79).
320	140		CHAR(8)	Reserved_25	Reserved.
328	146		BINARY(4)	Qdbfocs	Offset from the start of the FDT header, Qdb_Qdbfh, to the alternative collating sequence table section, Qdb_Qdbfacs.
332	14C		CHAR(4)	Reserved_26	Reserved.

Offset		Bit	Type	Field	Description
Dec	Hex				
336	150		CHAR(2)	Qdbfpact	Access path type. *AR* Arrival sequence access path. *KC* Keyed sequence access path with duplicate keys allowed. Duplicate keys are accessed in first-changed-first-out (FCFO) order. *KF* Keyed sequence access path with duplicate keys allowed. Duplicate keys are accessed in first-in-first-out (FIFO) order. *KL* Keyed sequence access path with duplicate keys allowed. Duplicate keys are accessed in last-in-first-out (LIFO) order. *KN* Keyed sequence access path with duplicate keys allowed. No order is guaranteed when accessing duplicate keys. *KU* Keyed sequence access path with no duplicate keys allowed (UNIQUE). *EV* Encoded vector with a 1-, 2-, or 4-byte vector.
338	152		CHAR(6)	Qdbfhrls	File version, release, and modification level. VxRyMz, where x is the version, y the release, and z the modification level.
344	158		CHAR(20)	Reserved_27	Reserved.
364	16C		BINARY(4)	Qdbpfof	Offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the physical file specific attributes section, Qdb_Qdbfphys (page 63).
368	170		BINARY(4)	Qdblfof	Offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the logical file specific attributes section, Qdb_Qdbflogl (page 69).
372	174		CHAR(6)	Qdbfssfp	Sort sequence table.
372	174		CHAR(1)	Qdbfnlsb	Flags.
372	174	0	BIT(3)	Qdbfsscs	Sort sequence table (SRTSEQ) indicators. *B'000'* No sort sequence table for the file; however, an alternate collating sequence table was specified. *B'010'* No sort sequence table is used for the file, and the hexadecimal value of the characters will be used to determine the sort sequence (*HEX). *B'100'* A sort sequence table was specified for the file.
372	174	3	BIT(5)	Reserved_103	Reserved.
373	175		CHAR(3)	Qdbflang	Language identifier (LANGID).
376	178		CHAR(2)	Qdbfcnty	Country or region identifier (CNTRYID).
378	17A		BINARY(4)	Qdbfjorn	Offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the journal section, Qdb_Qdbfjoal (page 84).
382	17E		BINARY(4)	Qdbfevid	Initial number of distinct values an encoded vector access path was allowed at creation.
386	182		CHAR(14)	Reserved_28	Reserved.

Physical File Specific Attributes (Qdb_Qdbfphys)

You can locate the *Qdb_Qdbfphys* section with the offset Qdbpfof (page 62), in the FDT header section.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		CHAR(2)	Qdbfpalc	Allocate/contiguous storage (ALLOCATE and CONTIG) *DN* New members added to the file allow the system to determine storage space that is allocated for the member (ALLOCATE(*NO)) *IC* New members added to file use the initial number of records to determine storage space that is allocated for the member (ALLOCATE(*YES)) and the storage attempted to be allocated contiguously (CONTIG(*YES)). *IN* New members added to file use the initial number of records to determine storage space that is allocated for the member (ALLOCATE(*YES)) and storage is not attempted to be allocated contiguously (CONTIG(*YES)).
2	2		CHAR(1)	Qdbfcmps	Maximum percentage of deleted records allowed (DLTPCT). *X'00'* The number of deleted records is not checked when the member is closed (*NONE). *X'01' through X'64'* The largest percentage of deleted records the member should have.
» 3	3		BINARY(4)	Qdbfoff_sqpt	For partitioned tables, offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the Partitioned Table area, Qdb_Qdbfsqpt_Head. «
7	7		CHAR(4)	Reserved_29	Reserved.
11	B		BINARY(4)	Qdbfprnum	Initial number of records (SIZE). *0* The number of records that can be inserted into each member is not limited by the user. The system determines the maximum member size (*NOMAX) *1 through 2,147,483,646* The number of records that can be inserted before an automatic extension occurs.
15	F		BINARY(2)	Qdbfpri	Increment number of records (SIZE). *0 through 32,767* The maximum number of records that can inserted into the member after an automatic extension occurs.
17	11		BINARY(2)	Qdbfprinum	Maximum number of increments (SIZE). *0 through 32,767* The maximum number of increments that can be automatically added to the member.
19	13		BINARY(4)	Qdbforid	Offset from the start of FDT header, Qdb_Qdbfh (Qdb_Qdbfh (page 56)), to the Record ID Codes for program described physical files, Qdbforid.
23	17		CHAR(1)	Qflags	Flags.

Offset					
Dec	Hex	Bit	Type	Field	Description
23	17	0	BIT(1)	Qdbfrdel	Reuse deleted records (RESUEDLT). If on, deleted member record space is reused by the system on write (insert) requests (*YES).
23	17	1	BIT(3)	Reserved_30	Reserved.
23	17	4	BIT(1)	Qdbfsqlt	SQL table indicator. If on, the file is a SQL table.
» 23	17	5	BIT(1)	Qdbfmqt	SQL materialized query table indicator. If on, the file is a SQL materialized query table.
23	17	6	BIT(1)	Qdbfsqpt	Partitioned table indicator. If on, the file is a partitioned table.
23	17	7	BIT(1)	Reserved_31	Reserved. «
24	18		BINARY(4)	Qdbfotrg	Offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the trigger description area, Qdbftrg (page 64).
28	1C		BINARY(2)	Qdbftrgn	Number of triggers.
30	1E		BINARY(4)	Qdbfofcs	Offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the constraint definition area, Qdbf_Constraint (page 65).
34	22		BINARY(4)	Qdbfcstn	Number of constraints for the file.
38	26		BINARY(4)	Qdbfodl	Offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the datalinks area, Qdb_Qdbfdtalnk (page 68).
» 42	2A		BINARY(4)	Qdbfovw_mqt	For SQL materialized query tables, offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the SQL area, Qdb_Qdbfv.
46	2E		CHAR(2)	Reserved_32	Reserved. «

Trigger Description Area (Qdb_Qdbftrg)

You can locate the *Qdb_Qdbftrg* section with the offset Qdbfotrg (page 64) in the Physical File Specific Attributes section. This section is repeated by the number of triggers, Qdbftrgn.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		CHAR(1)	Qdbftrgt	Trigger time. *1* Run the trigger after the change operation. *2* Run the trigger before the operation.
1	1		CHAR(1)	Qdbftrge	Trigger event. *1* An insert operation. *2* A delete operation. *3* An update operation. *4* A read operation.
2	2		CHAR(10)	Qdbftpgm	Trigger program name.
12	C		CHAR(10)	Qdbftplb	Trigger program library name.

Offset		Bit	Type	Field	Description
Dec	Hex				
22	16		CHAR(1)	Qdbftupd	Trigger update condition. *1* Always call the trigger program when updating the file. *2* Call the trigger program only when the updated values are changed. This field is ignored for insert and delete operations.
23	17		CHAR(1)	Qdbftrgf	Trigger flags.
23	17	0	BIT(1)	Qdbfalrc	Allow repeated change indicator. If on, repeated changes are allowed.
23	17	1	BIT(2)	Qdbfttths	Trigger threadsafe indicator. *B'00'* Not known. *B'10'* Not threadsafe. *B'11'* Threadsafe.
23	17	3	BIT(2)	Qdbftmta	Multithreaded job action indicator. *B'01'* Run, send diagnostic. *B'10'* Do not run, send escape. *B'11'* Run, do not send message.
23	17	5	BIT(1)	Qdbftqmt	QMLTTHDACN system value use. If on, the system value was used to determine Qdbftmta.
23	17	6	BIT(1)	Qdbf_more_trg_info	Whether more trigger information is available if format FILD0400 is requested. *B'0'* No more trigger information is available. *B'1'* More trigger information is available.
23	17	7	BIT(1)	Reserved_200	Reserved.

Constraint Definition Header (Qdb_Qdbf_Constraint)

You can locate the *Qdb_Qdbf_Constraint* section with the offset Qdbfofcs located in the physical file specific attributes section, Qdb_Qdbfphys.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		BINARY(4)	Qdbf_csto	Offset from Qdbf_Constraint to the next section for this constraint.
4	4		BINARY(4)	Qdbf_hlen	Constraint entry header length in bytes.
8	8		CHAR(1)	Qdbf_type	Constraint type (TYPE) *F* Referential constraint *P* Primary unique constraint *U* Unique constraint. *C* Check constraint.

Offset		Bit	Type	Field	Description
Dec	Hex				
9	9		CHAR(1)	Qdbf_chkpd	Check pending attribute. *N* The constraint is not in check pending. *Y* The constraint is in check pending.
10	A		CHAR(1)	Qdbf_state	Constraint state. *D* The constraint is defined. *E* The constraint is established.
11	B		CHAR(1)	Qdbf_abled	Constraint enablement. *D* The constraint is disabled. *E* The constraint is enabled.
12	C		CHAR(13)	Qdbf_add_ts	Constraint date. The date is in the internal standard format (ISF), CYYMMDDHHMMSS.
25	19		CHAR(10)	Qdbf_cst_lin	Constraint library name.
35	23		BINARY(4)	Qdbf_cst_lp2	Constraint name (delimited) length
39	27		CHAR(25)	Reserved_54	Reserved.
64	40		CHAR(258)	Qdbf_cst_name	Constraint name (CST).

Constraint Definition Entries

The number of constraint definition entries depends on the type of constraint.

- A referential constraint, type F, has three structures in this sequence:
  1. Qdbf_Keyn for parent file
  2. Qdbf_Keyn for dependent file
  3. Qdbf_Riafk_Afkd
- A unique constraint, type U, has one Qdbf_Keyn structure.
- A primary unique constraint, type P, has one Qdbf_Keyn structure.
- A check constraint, type C, has one Qdbf_Chk_Cst structure.

Constraint Keys (Qdb_Qdbf_Keyn)

The *Qdb_Qdbf_Keyn* section is located with the offset Qdbf_Hlen in the constraint definition header, Qdbf_Constraint (page 65). When the constraint is referential constraint, the offset to the next section is located with the offset Qdbf_Kslen in this structure.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		BINARY(4)	Qdbf_kslen	Constraint key structure length. The length, in bytes, of this constraint key structure. This is also the offset to from Qdbf_Keyn (page 66) to the next structure for this constraint.
4	4		BINARY(4)	Qdbf_nokys	Number of keys, 1 through 120. The number of key fields for the constraint key.
8	8		BINARY(4)	Qdbf_klen	Constraint key length.
12	C		CHAR(52)	Revcst_7	Reserved.
64	40		Array of CHAR(32)	Qdbf_narray	Key name array.

Key Name Array (Qdb_Qdbf_Narray)

This array follows the constraint keys structure, Qdbf_Keyn (page 66). The number of constraint key name array entries is in field Qdbf_nokys in the constraint keys structure.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		CHAR(10)	Qdbf_kname	Key name (PRNKEY KEY)
10	A		CHAR(22)	Revcst_6	Reserved.

Referential Constraint Definition (Qdb_Qdbf_Riafk_Afkd)

You can locate this section with the offset Qdbf_kslen in the constraint keys structure, Qdbf_Keyn (page 66), that precedes this structure. This structure exists only if the constraint is a referential constraint.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		CHAR(64)		Parent file (PRNFILE).
0	0		CHAR(10)	Qdbf_riafk_pkfn	Parent file name.
10	A		CHAR(10)	Qdbf_riafk_pkln	Parent file library name.
20	14		CHAR(44)	Revcst_3	Reserved.
64	40		CHAR(1)	Qdbf_riafk_fkcdr	Delete rule (DLTRULE). *C* *CASCADE *D* *SETDFT *L* *SETNULL *N* *NOACTION (default value) *R* *RESTRICT
65	41		CHAR(1)	Revcst_4	Reserved
66	42		CHAR(1)	Qdbf_riafk_fkcur	Update rule (UPDRULE) *N* *NOACTION (default value) *R* *RESTRICT
67	43		CHAR(61)	Revcst_5	Reserved.

Check Constraint (Qdb_Qdbf_Chk_Cst)

This section is located with the offset Qdbf_Hlen in the constraint definition header, Qdbf_Constraint. This structure exists only if the constraint is a check constraint.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		BINARY(4)	Qdbf_chkcst_len	Check constraint structure length. The length, in bytes, of the check constraint structure Qdb_Qdbf_Chk_Cst.
4	4		BINARY(4)	Qdbf_chkexpr_len	Check constraint expression length. The length of the check constraint expression Qdbf_chkexpr.
8	8		CHAR(24)	Revcst_8	Reserved.

Offset		Bit	Type	Field	Description
Dec	Hex				
32	20		CHAR(*)	Qdbf_chkexpr	Check constraint expression.

Datalink Header (Qdb_Qdbfdtalnk)

The *Qdb_Qdbfdtalnk* section is the header for the datalink columns that have linked servers. There will be one header and one or more datalink column entries defined by the Qdb_Qdbfdlcole (page 68) structure. You can locate this structure with the offset Qdbfodl in the Physical File Specific Attributes structure, Qdb_Qdbfphys (page 63).

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		BINARY(4)	Qdbfdlcoln	Number of datalink columns with links to servers.
4	4		BINARY(4)	Qdbfdlocole	Offset from the start of Qdb_Qdbfdtalnk to the first datalink column entry, (Qdb_Qdbfdlcole (page 68)).
8	8		CHAR(1)	Qdbfdllnkp	Link pending attribute. **Link pending** is a state that indicates to the user the file has one or more datalink field values (under the file link control attribute) where the system does not know whether or not the field is really linked to a file on the DataLink File Manager server. The **Datalink File Manager** is a function that tracks which files are linked to a specific database file. *N* The file is not in link pending. *Y* The file is in link pending.
9	9		CHAR(23)	Revdl_1	Reserved.

Datalink Column Entry (Qdb_Qdbfdlcole)

The *Qdb_Qdbfdlcole* section repeats for the number of columns (Qdbfdlcoln) defined in structure Qdb_Qdbfdtalnk (page 68). You can locate the first column entry using offset Qdbfdlocole in structure Qdb_Qdbfdtalnk. Since Qdb_Qdbfdlcole is a varying length structure, use length Qdbfdlcelen to get to the next column entry.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		BINARY(4)	Qdbfdlcelen	Length of this datalink column entry. Use this length to get to the next datalink column entry.
4	4		BINARY(4)	Qdbfdlsevn	Number of servers linked for this column.
8	8		CHAR(10)	Qdbfdlcolnm	Column name.
18	12		CHAR(14)	Revdl_2	Reserved.
32	20		Array of CHAR(254)	Qdbfdlsevnm	Array of server names linked to the datalink column. The number of array entries is defined by Qdbfdlsevn.

Record ID Codes (Qdb_Qdbfdrtb)

The *Qdb_Qdbfdrtb* section describes the record ID codes for program described physical files. The record ID code information is an array with variable length entries. You can locate this section with the offset Qdbforid located in the physical file specific attributes section, Qdb_Qdbfphys (page 63).

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		CHAR(16)		Record ID code header.
0	0		BINARY(2)	Qdbfdrnum	Number of record ID code array entries, 0 through 70.
2	2		BINARY(4)	Qdbfdrtl	Size of this record ID code table in bytes, 0 through 256.
6	6		CHAR(10)	Reserved_33	Reserved.
16	10		Array of CHAR(32)	Qdbfdrae	Record ID code array entry.

Record ID Codes Array (Qdb_Qdbfdrae)

This array follows the record ID codes structure, (Qdb_Qdbfdrtb (page 68)). The number of record ID code array entries is in Qdbfdrnum.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		CHAR(4)	Reserved_34	Reserved.
4	4		CHAR(10)	Qdbfdrnm	External name.
14	E		BINARY(2)	Qdbfdrrp	Relative field position, 1 through 8000. The relative position of the field in the record format.
16	10		CHAR(2)	Qdbfdrco	Comparison operator. *EQ* Compare equal. *NE* Compare not equal. *ZN* Compare zone. *NZ* Compare not zone. *DG* Compare digit. *ND* Compare not digit.
18	12		BINARY(2)	Qdbfdrln	Length of test value. Test value length must be 1.
20	14		CHAR(1)	Qdbfdrtv	Test value.
21	15		CHAR(1)	Qdbfdrao	AND/OR/last operator. *0* Last operator entry. *1* AND with next array entry. *2* OR with next array entry.
22	16		CHAR(10)	Reserved_35	Reserved.

Logical File Specific Attributes (Qdb_Qdbflogl)

You can locate the *Qdb_Qdbflogl* section with the offset Qdbflfof located in the FDT header section, Qdb_Qdbfh.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		BINARY(4)	Qdbfoj	Offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the join specifications, Qdbfj (page 70).

Offset					
Dec	Hex	Bit	Type	Field	Description
4	4		BINARY(2)	Qdbfscsn	Total number of select/omit statements for all record formats, 1 through 32,767.
6	6		CHAR(10)	Qdbflxp	Record format selector program (FMTSLR) *X'00'* No record format selector program (*NONE).
16	10		CHAR(10)	Qdbflxl	Record format selector program library (FMTSLR) *X'00'* No record format selector program (*NONE).
26	1A		BINARY(4)	Qdbfovw	Offset from the start of the FDT header, Qdb_Qdbfh (page 70), » to the SQL area, « Qdb_Qdbfv.
30	1E		CHAR(1)	Qlfa	Logical file attributes
30	1E	0	BIT(2)	Reserved_36	Reserved.
30	1E	2	BIT(1)	Qdbfjoin	Join logical file indicator (JFILE). If on, the file is a join logical file.
30	1E	3	BIT(1)	Qdbfdyns	Dynamic selection indicator (DYNSLT). If on, the selection and omission tests specified for the file are done when the file is read. If off, when the access path is updated.
30	1E	4	BIT(1)	Qdbfsqlv	SQL view indicator. If on, the file is an SQL view.
30	1E	5	BIT(1)	Qdbfsqli	SQL index indicator. If on, the file is an SQL index.
30	1E	6	BIT(2)	Reserved_37	Reserved.
31	1F		CHAR(1)	Qdbfjtyp	Join file type. *I* An inner join. Default entries are not supplied if a join value does not exist. *P* A partial outer join. Default values are supplied if a join value does not exist.
32	20		BINARY(2)	Qdbfsrcd	Coded character set identifier (CCSID) for select/omit constants. *0* There are no select/omit constants for the file. *1 through 65,535* The CCSID.
34	22		CHAR(1)	Qdbfwchk	With check option. *C* The with-check option was specified with cascade. *L* The with-check option was specified with local. *N* No with-check option was specified. The value N is set for all logical files. The values C and L only apply to SQL views.
35	23		CHAR(13)	Reserved_38	Reserved.

Join Specifications (Qdb_Qdbfj)

The join specifications, *Qdb_Qdbfj*, are a linked list. There is an entry in the linked list for each join to-file. Each entry defines the join logical file's based on physical files and the fields in the from-file and the to-file used to join the based on physical file.

You can locate this section with the offset Qdbfoj located in the FDT header section, Qdb_Qdbfh (page 56).

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		BINARY(4)	Qdbfjnho	Offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the join specifications, Qdbfj (page 70), for the next join to-file.
4	4		CHAR(4)	Reserved_40	Reserved.
8	8		BINARY(2)	Qdbfjknum	Number of join field specifications (JFLD), 1 through 32,767.
10	A		BINARY(2)	Qdbfjdnum	Number of join duplicate sequence specifications (JDUPSEQ), 1 through 32,767.
12	C		BINARY(2)	Qdbfjffnum	Join from-file number (JOIN), 1 through » 255. « This number indicates which based on physical file to join the to-file from.
14	E		BINARY(2)	Qdbfjtfnum	Join to-file number (JOIN), 2 through » 256. « This number indicates which based on physical to-file this join specification relates to.
16	10		CHAR(24)	Reserved_41	Reserved.
40	28		BINARY(4)	Qdbfjsao	Offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the join specification array, Qdb_Qdbfjfld (page 71)
44	2C		BINARY(4)	Qdbfjdao	Offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the join duplicate sequence array, Qdb_Qdbfjdup (page 72), for this join to-file.

Join Specification Array (Qdb_Qdbfjfld)

You can locate the *Qdb_Qdbfjfld* section with the offset Qdbfjsao (page 71) located in the join header section, Qdb_Qdbfj. The number of join specification array entries may be up to one less than the number of data members, Qdbflbnum (page 56) , located in the FDT header section, Qdb_Qdbfh.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		CHAR(10)	Qdbfjfnm	Join from-field-name (JFLD)
10	A		BINARY(2)	Qdbfjfnum	Join from-field reference number. *0* Join from-field is a field in the join logical file's record format. **1 through » 255 «** The number of the base on physical from-file corresponding with its position in the JFILE statement that contains this join from-field.
12	C		CHAR(2)	Reserved_42	Reserved.
14	E		CHAR(2)	Qdbfjop	Join operation. This is always set to 'EQ'.
16	10		CHAR(10)	Qdbfjtnm	Join to-field name (JFLD).
26	1A		BINARY(2)	Qdbfjtnum	Join to-field reference number. *0* The join to-field is a field in the logical file's record format. **2 through » 256 «** The number of the based on physical to-file corresponding with its position in the JFILE statement that contains this join to-field.
28	1C		CHAR(20)	Reserved_43	Reserved.

Join Duplicate Sequence Specification Array (Qdb_Qdbfjdup)

You can locate the *Qdb_Qdbfjdup* section with the offset Qdbfjdao (page 71) in the join section, Qdb_Qdbfj. The number of join specification array entries may be up to one less than the number of data members, Qdbflbnum (page 56), located in the FDT header section, Qdb_Qdbfh.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		CHAR(10)	Qdbfjdnm	Join specification field name (JDUPSEQ).
10	A		BINARY(2)	Qdbfjdjnum	Join sequence field name reference number. *0* The join sequencing field name is a file in the join logical file's record format. **2 through 256** The number of the based on physical to-file corresponding with its position in the JFILE statement that contains this sequencing field name.
12	C		CHAR(1)	Qjsfna	Join sequencing field name attributes.
12	C	0	BIT(1)	Qdbfjdd	Ascending/descending sequence indicator. If on, indicates a descending field (*DESCEND).
12	C	1	BIT(7)	Reserved_44	Reserved.
13	D		CHAR(19)	Reserved_45	Reserved.

SQL Area (Qdb_Qdbfv)

The SQL area, *Qdb_Qdbfv*, contains the SQL select statement. For logical files, you can locate this section with the offset Qdbfovw located in the logical file specific attributes section, (Qdb_Qdbflogl (page 69)). For SQL materialized query tables, you can locate this section with the offset Qdbfovw_mqt located in the physical file specific attributes section, (Qdb_Qdbfphys (page 63)).

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		CHAR(56)		SQL area header.
0	0		CHAR(16)	Reserved_39	Reserved.
16	10		BINARY(4)	Qdbfvs_start	Starting offset of the SELECT statement within the CREATE TABLE statement.
20	14		BINARY(4)	Qdbfvs_end	Ending offset of the SELECT statement within the CREATE TABLE statement.
24	18		BINARY(2)	Qdbfvs_ccsid	Coded Character Set Identifier, CCSID, for the Select Statement. Views created prior to V5R1M0 will return a CCSID value of 0.
26	1A		BINARY(4)	Qdbfmqtd_o	For SQL materialized query tables, offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the SQL materialized query table dependency area, Qdb_Qdbfmqtd_Head.

Offset		Bit	Type	Field	Description
Dec	Hex				
30	1E		CHAR(2)	Qdbfmqt_cmt	The isolation level in effect for the SQL materialized query table: *NC* *NONE no isolation *UR* *CHG uncommitted read. *CS* *CS cursor stability *CL* *CS with keep locks *RS* *ALL read stability *RR* *RR repeatable read
32	20		CHAR(1)	Qdbfvflgs	Flags.
32	20	0	BIT(1)	Qdbfmqt_init	SQL materialized query table initial data. If on, data is inserted into the table immediately as part of the SQL Create Table statement. If off, data is deferred until the SQL Refresh Table statement is issued for the table.
32	20	1	BIT(1)	Qdbfmqt_ maint	SQL materialized query table maintenance. If on, table is user-maintained.
32	20	2	BIT(1)	Qdbfmqt_ refresh	SQL materialized query table refresh. If off, table is refresh-deferred.
32	20	3	BIT(1)	Qdbfmqt_opt	SQL materialized query table optimization. If on, table is disabled for optimization. If off, table is enabled for optimization.
32	20	4	BIT(4)	Reserved_88	Reserved.
33	21		CHAR(23)	Reserved_89	Reserved.
56	38				SQL select statement structure
56	38		BINARY(4)	Qdbfvssl	Select statement length.
60	3C		CHAR(*)	Qdbfvsst	SQL select statement.

SQL Materialized Query Table Dependency Header (Qdb_Qdbfmqtd_Head)

The *Qdb_Qdbfmqtd_Head* section is located with the offset Qdbfmqtd_o in the SQL area, *Qdb_Qdbfv*. This structure exists only if the file is a SQL materialized query table.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		BINARY(4)	Qdbfmqtd_len_ sum	Total length, in bytes, of the dependency area. This includes both the dependency header and the dependency array.
4	4		BINARY(4)	Qdbfmqtd_#_deps	Number of file entries in the dependency array.
8	8		CHAR(56)	Reserved_16	Reserved.
64	40		CHAR(*)		Dependency array, Qdb_Qdbfmqtd, for this SQL materialized query table.

SQL Materialized Query Table Dependency Entry (Qdb_Qdbfmqtd)

The *Qdb_Qdbfmqtd* section repeats in the dependency array for the number of depended-on files (Qdbfmqtd_#_deps) defined in structure *Qdb_Qdbfmqtd_Head*. The first dependency entry follows the dependency header *Qdb_Qdbfmqtd_Head*. Use length Qdbfmqtd_len to get to the next dependency entry.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Field**	**Description**
0	0		CHAR(16)	Reserved_117	Reserved.
16	10		BINARY(4)	Qdbfmqtd_len	Length, in bytes, of this dependency entry. Use this length to get to the next dependency entry.
20	14		CHAR(10)	Qdbfmqtd_file	Name of the file that the materialized query table is dependent on. The file is specified in the select-statement of the materialized query table.
30	1E		CHAR(10)	Qdbfmqtd_lib	Name of the library that the depended-on file resides in.
40	28		CHAR(56)	Reserved_116	Reserved. «

Partitioned Table Header (Qdb_Qdbfsqpt_Head)

The *Qdb_Qdbfsqpt_Head* section is located with the offset Qdbfoff_sqpt. This structure exists only if the file is a partitioned table.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Field**	**Description**
0	0		BINARY(4)	Qdbfsqpt_Len_ Sum	Total length, in bytes, of the header area. This includes all the partition areas and partition key areas.
4	4		CHAR(4)	Qdbfsqpt_Sql_ Reserved4	Reserved.
8	8		CHAR(1)	Qdbfsqpt_Ran_ Hh	Partitioning is range or hash. *R* Range Partitioning *H* Hash Partitioning
9	9		CHAR(1)	Qdbfsqpt_ Lfld	Field has long field name. *Y* Field has a long name. *N* Field does not have a long name.
10	0A		CHAR(230)	Qdbfsqpt_ Reserved230	Reserved.
240	F0		BINARY(4)	Qdbfsqpt_Num_ Parts	Number of file partitions.
244	F4		BINARY(4)	Qdbfsqpt_Part_ Offset	The offset to the first partition. This is from the start of the Partitioned Table Header, Qdb_Qdbfsqpt_Head (page "Partitioned Table Header (Qdb_Qdbfsqpt_Head)"), to the first partition, Qdb_Qdbfsqpt_Part (page "Partitioned Area (Qdb_Qdbfsqpt_Part)" on page 75).
248	F8		BINARY(4)	Qdbfsqpt_Hk_ Offset	The offset to the hash key area. This offset is only set if hash partitioning is being done for the partitioned table. This is from the start of the Partitioned Table Header, Qdb_Qdbfsqpt_Head, to the partition hash header Qdb_Qdbfsqpt_Head_Hash (page "Partition Header Hash Key Area (Qdb_Qdbfsqpt_Head_Hash)" on page 77), if hash partitioning is being done.

Offset		Bit	Type	Field	Description
Dec	Hex				
252	FC		CHAR(100)	Qdbfsqpt_ Reserved_100	Reserved.
352	160		CHAR(*)		Partitioned area, Qdb_Qdbfsqpt_Part (page "Partitioned Area (Qdb_Qdbfsqpt_Part)"), for this partitioned table and partition hash key area Qdb_Qdbfsqpt_Head_Hash (page "Partition Header Hash Key Area (Qdb_Qdbfsqpt_Head_Hash)" on page 77), if hash partitioning is being done. ≪

Partitioned Area (Qdb_Qdbfsqpt_Part)

The *Qdb_Qdbfsqpt_Part* section is located with the offset Qdbfsqpt_Part_Offset. This structure exists only if the file is a partitioned table.The repeating structure(Qdb_Qdbfsqpt_Part) can be located with the Qdb_Qdbfsqpt_Part_Len value.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		BINARY(4)	Qdbfsqpt_Part_ Len	Total length, in bytes, of this partition area. This includes all other partition key areas. Use this length to get to the next partition area.
4	4		BINARY(4)	Qdbfsqpt_Part_ DSN	Partition data space number.
8	8		BINARY(4)	Qdbfsqpt_Part_ Num	Partition number.
12	0C		BINARY(4)	Qdbfsqpt_Pname_L en	The length of the partition name.
16	10		CHAR(128)	Qdbfsqpt_ Pname	The partition name.
144	90		CHAR(80)	Qdbfsqpt_Mbr_ Reserved80	Reserved.
224	E0		BINARY(4)	Qdbfsqpt_Num_ Keys	The number of partition range keys.
228	E4		BINARY(4)	Qdbfsqpt_Range_ Offset	The offset to the first partition range key area for this partition. If hash partitioning is being done, this value will not be set. This is from the start of the Partitioned Area, Qdb_Qdbfsqpt_Part (page "Partitioned Area (Qdb_Qdbfsqpt_Part)"), to the Qdb_Qdbfsqpt_Range (page 75).
232	E8		CHAR(8)	Qdbfsqpt_ Reserved_8	Reserved.
240	F0		CHAR(*)		Partition range key area Qdb_Qdbfsqpt_Range (page "Partition Range Key Area (Qdb_Qdbfsqpt_Range)"), for this partitioned table. This entry repeats for the number of partition range keys. ≪

Partition Range Key Area (Qdb_Qdbfsqpt_Range)

The *Qdb_Qdbfsqpt_Range* section is located with the offset Qdbfsqpt_Ran_Offset. This structure exists only if the file is a partitioned table that has range partitioning. Use Qdbfsqpt_Range_Len to get to the next entry.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		BINARY(4)	Qdbfsqpt_Range_ Len	Total length, in bytes, of this partition range key area.
4	4		CHAR(12)	Qdbfsqpt_ Reserved12	Reserved.
16	10		CHAR(10)	Qdbfsqpt_Range_ Key	Partition range key name.
26	1A		BINARY(2)	Qdbfsqpt_Range_ Reserved	Reserved.
28	1C		CHAR(1)	Qdbfsqpt_ Rmin	Partition range key start value minimum. *Y* Yes *N* No
29	1D		CHAR(1)	Qdbfsqpt_ Rmax	Partition range key end value maximum. *Y* Yes *N* No
30	1E		CHAR(1)	Qdbfsqpt_ Rsin	Partition range key start value inclusive. *Y* Yes *N* No
31	1F		CHAR(1)	Qdbfsqpt_ Rein	Partition range key end value inclusive. *Y* Yes *N* No
32	20		CHAR(15)	Qdbfsqpt_ reserved15	Reserved.
47	2F		CHAR(1)	Qdbfsqpt_ Rnull	Partition range key value can include nulls. *Y* Yes *N* No
48	30		BINARY(4)	Qdbfsqpt_ Rslen	Length in bytes of the starting partition range key value string.
52	34		BINARY(4)	Qdbfsqpt_ Relen	Length in bytes of the ending partition range key value string.
56	38		BINARY(4)	Qdbfsqpt_Rstart_ Offset	The offset to the starting partition range key value string. This is from the start of Qdb_Qdbfsqpt_Range (page "Partition Range Key Area (Qdb_Qdbfsqpt_Range)" on page 75).
60	3C		BINARY(4)	Qdbfsqpt_Rend_ Offset	The offset to the ending partition range key value string. This is from the start of Qdb_Qdbfsqpt_Range (page "Partition Range Key Area (Qdb_Qdbfsqpt_Range)" on page 75).
64	40		CHAR(14)	Qdbfsqpt_ Reserved14	Reserved.
78	4E		BINARY(2)	Qdbfsqpt_ Rccsid	Coded character set identifier (CCSID) for the start and end strings.
80	50		CHAR(*)		Start value string and end value string for this partition range key. «

Partition Header Hash Key Area (Qdb_Qdbfsqpt_Head_Hash)

The *Qdb_Qdbfsqpt_Head_Hash* is the header area for the hash keys. It is located with the offset Qdbfsqpt_Hk_Offset. This structure exists only if the file is a partitioned table that has hash partitioning.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		BINARY(4)	Qdbfsqpt_Head _Hash_Len	Length, in bytes, of the partition header hash key area.This includes the total length of all hash key entries.
4	4		BINARY(4)	Qdbfsqpt_Head _Num_Keys	Total number of hash keys.
8	8		CHAR(56)	Head_ Reserved_56	Reserved.
64	40		Array of CHAR(32)	Qdbfsqpt_ hash	Key name array. «

Partition Hash Key Area (Qdb_Qdbfsqpt_Hash)

The *Qdb_Qdbfsqpt_Hash* section repeats for the number of partition hash keys,Qdbfsqpt_Head_Num_Keys. It follows Qdb_Qdbfsqpt_Head_Hash (page "Partition Header Hash Key Area (Qdb_Qdbfsqpt_Head_Hash)").

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		CHAR(10)	Qdbfsqpt_ Hash_Key	Partition hash key name.
10	0A		CHAR(22)	Hash_ Reserved_22	Reserved. «

Alternative Collating Sequence Table (Qdb_Qdbfacs)

You can locate this section with the offset Qdbfocs (page 61) in the FDT header section, Qdb_Qdbfh. This section is also referred to as the Sort Sequence Table. A sort sequence table can be either single-byte or UCS-2. If the UCS-2 table length, Qdbf_UCS2_Srtseq_Len, is non-zero, then it is a UCS-2 sort sequence table and the single-byte table, Qdbfacst, will be cleared.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		CHAR(256)	Qdbfacst	Alternative collating sequence table or single-byte sort sequence (ALTSEQ/STRSEQ) table.
256	100		BINARY(2)	Qdbfccsd	Coded character set identifier (CCSID) for the single-byte table.
258	102		CHAR(20)	qdbfsrts	Sort sequence table.
258	102		CHAR(10)	Qdbftbln	Sort sequence table name.
268	10C		CHAR(10)	Qdbftbll	Sort sequence table library name.
278	116		CHAR(1)	Qdbfsrtf	Sort sequence table attributes.
278	116	0	BIT(1)	Qdbfwght	Sort sequence table weight indicator for the single-byte table. If on, indicates the sort sequence table is unique weighted. If off, it is share weighted.

Offset		Bit	Type	Field	Description
Dec	Hex				
278	116	1	BIT(1)	Qdbfsubc	Sort sequence table substitution character indicator for the single-byte table. If on, indicates the sort sequence table has substitution character.
278	116	2	BIT(1)	Qdbf_UCS2_ Wght	Sort sequence table weight indicator for the UCS-2 table. If on, indicates the sort sequence table is unique weighted. If off, it is share weighted.
278	116	3	BIT(5)	Reserved_104	Reserved.
279	117		BINARY(4)	`Qdbf_UCS2 _Srtseq_Len`	Length of the UCS-2 sort sequence table, Qdbf_UCS2_Srtseq, in bytes.
283	11B		BINARY(2)	Qdbf_UCS2_ Ccsd	Coded character set identifier (CCSID) for the UCS-2 table.
285	11D		CHAR(19)	Reserved_101	Reserved.
304	130		CHAR(*)	`Qdbf_UCS2 _Srtseq`	UCS-2 sort sequence table. The table exists if the length, Qdbf_UCS2_Srtseq_Len, is greater than zero.

IDDU/SQL Data Dictionary Area (Qdb_Qdbfdic)

You can locate the *Qdb_Qdbfdic* section with offset Qdbfodic (page 61) in the FDT header section, Qdb_Qdbfh.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		CHAR(1)	Qdbfdilk	Data dictionary link status. *L* The file is linked to the a data dictionary. *U* The file is not linked to the a data dictionary.
1	1		CHAR(10)	Qdbfinm	Data dictionary library name.
11	B		CHAR(10)	Qdbfifd	Data dictionary file definition name.
21	15		CHAR(11)	Qdbfdiid	Data dictionary internal file definition identifier. This field maps to ZONED(11,0).
32	20		CHAR(4)	Reserved_46	Reserved.
36	24		BINARY(4)	Qdbfdicl	Data dictionary file definition comment length.
40	28		BINARY(2)	Qdbfdicc	Data dictionary file definition comment CCSID. *0* There is no comment for the file. *1 through 65,535* The CCSID of the comment.
42	2A		BINARY(4)	Qdbfolng	Offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the SQL long/alias file names area, Qdb_Qdbflngn (page 79).
46	2E		BINARY(2)	Qdbflnnum	Number of long/alias file names for the file.
48	30		CHAR(16)	Reserved_47	Reserved.
64	40		CHAR(*)	Qdbfdict	Data dictionary file definition comment text.

SQL Long/Alias File Name Area (Qdb_Qdbflngn)

The SQL long/alias file name area contains the files alternate names that can be used to access the file when using the system's SQL interfaces. You can locate the *Qdb_Qdbflngn* section with the offset Qdbfolng (page 78) in the IDDU/SQL data dictionary section.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		BINARY(2)	Qdbflnel	Long/alias file name entry length in bytes. The length of this entry. This is also the offset from Qdbflnen to the next long/alias entry.
2	2		CHAR(1)	Qdbflnfl	Long/alias file name flags.
2	2	0	BIT(1)	Qdbflndl	Long/alias file name input delimited indicator. If on, indicates the long/alias file name was delimited when input.
2	2	1	BIT(7)	Reserved_111	Reserved
3	3		BINARY(2)	Qdbflnlg	Long/alias file name (non-delimited) length.
5	5		CHAR(11)	Reserved_112	Reserved.
16	10		CHAR(*)	Qdbflnam	Long/alias file name (non-delimited).

File Scope Array (Qdb_Qdbfb)

A file scope array, Qdb_Qdbfb, is present for all database files. The number of data members, Qdbflbnum (page 56), contains the number of file scope array entries. Each entry contains a based on physical file name and, optionally, a record format name.

Externally described physical files have one entry that names the physical file record format. The entry's file name portion is not used.

Program described physical files have one entry for each data dictionary record format. The entry names the data dictionary record format. The entry's file name portion is not used.

Non-join logical files have one entry for each based on physical file. The entry names the based on physical file and describes the logical file record format to use with that file.

Join logical files have one entry for each based on physical file. The entry names the based on physical file. Only the first entry describes the logical file record format.

SQL view logical files have one entry for each based on physical file. The entry names the based on physical file that will be either an externally described physical file or another view logical file. Only the first entry describes the logical file record format.

You can locate this section with the offset Qdbfos (page 61) in the FDT header section, Qdb_Qdbfh.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		CHAR(48)	Reserved_48	Reserved.
48	30		CHAR(10)	Qdbfbf	Based on physical file name.
58	3A		CHAR(10)	Qdbfbfl	Based on physical file library name.
68	44		CHAR(10)	Qdbft	Record format name.

Offset		Bit	Type	Field	Description
Dec	Hex				
78	4E		CHAR(37)	Reserved_49	Reserved.
115	73		BINARY(2)	Qdbfbgky	Record format generic key field count, 0 through 120. If this file has an arrival sequence access path, this field is not applicable.
117	75		CHAR(2)	Reserved_50	Reserved.
119	77		BINARY(2)	Qdbfblky	Record format maximum key length, 1 through 2000. If this file has an arrival sequence access path, this field is not applicable.
121	79		CHAR(2)	Reserved_51	Reserved.
123	7B		BINARY(2)	Qdbffogl	Record format generic key length, 1 through 2000. If this file has an arrival sequence access path, this field is not applicable.
125	7D		CHAR(3)	Reserved_52	Reserved.
128	80		BINARY(2)	Qdbfsoon	Number of select/omit statements, 1 through 32,767.
130	82		BINARY(4)	Qdbfsoof	Offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the select/omit specification array, Qdb_Qdbfss (page 80).
134	86		BINARY(4)	Qdbfksof	Offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the key specification array, Qdb_Qdbfk (page 82).
138	8A		BINARY(2)	Qdbfkyct	Record format full key field count, 0 through 120. If this file has an arrival sequence access path, this field is not applicable.
140	8C		BINARY(2)	Qdbfgenf	Generic key field count for all record formats with this record format name, 0 through 120. If this file has an arrival sequence access path, this field is not applicable.
142	8E		BINARY(4)	Qdbfodis	Offset from the start of the FDT header, Qdb_Qdbfh (page 56) to the distributed file definition section.
146	92		CHAR(14)	Reserved_53	Reserved.

Select/Omit Specification Array (Qdb_Qdbfss)

The select/omit specification array (*Qdb_Qdbfss*) entries describe the record format fields to which the select/omit statement refer.

Non-join logical files can have one select/omit specification array for each file scope array entry.

Join logical files can have only one select/omit specification array. The first scope array entry for the join logical file contains the offset to the select/omit specification array.

You can locate this section with the offset Qdbfsoof (page 80) in the scope array entry section.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		CHAR(2)	Reserved_54	Reserved.
2	2		CHAR(1)	Qdbfssso	Select/omit statement rule. *A* A select/omit ANDed statement. *O* A select/omit omit statement. *S* A select/omit select statement.

Offset		Bit	Type	Field	Description
Dec	Hex				
3	3		CHAR(2)	Qdbfssop	Select/omit statement comparison (ALL COMP VALUES) *AL* Statement comparison for all (ALL). *EQ* Statement comparison for equal to (COMP EQ). *GE* Statement comparison for greater than or equal to (COMP GE). *GT* Statement comparison for greater than (COMP GT). *LE* Statement comparison for less or equal to (COMP LE). *LT* Statement comparison for less than (COMP LT). *NE* Statement comparison for not equal to (COMP NE). *NG* Statement comparison for not greater than (COMP NG). *NL* Statement comparison for not less than (COMP NL). *VA* Statement comparison for values (VALUES).
5	5		CHAR(10)	Qdbfssfn	Select/omit statement field name.
15	F		BINARY(2)	Qdbfsspnum	Number of select/omit statement parameters, 1 through 32,767.
17	11		CHAR(1)	Qsosaf	Select/omit statement attribute flags.
17	11	0	BIT(7)	Reserved_55	Reserved.
17	11	7	BIT(1)	Qdbfssfi	Select/omit statement external or internal name indicator. If on, indicates the statement is field name is an external record format name.
18	12		BINARY(2)	Qdbfssfj	Select/omit statement join reference number (JREF), 1 through » 256. « If this is not a join logical file, this field is not applicable.
20	14		CHAR(8)	Reserved_56	Reserved.
28	1C		BINARY(4)	Qdbfsoso	Offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the select/omit parameters, Qdb_Qdbfsp (page 81), for this select/omit statement.

Select/Omit Parameters (Qdb_Qdbfsp)

The *Qdb_Qdbfsp* section is a linked list of parameter descriptions. It describes the parameter values for this particular select/omit statement. The parameters are either a compare value or another record format field.

You can locate this section with the offset Qdbfsoso (page 81) in the select/omit array section, Qdb_Qdbfss.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		BINARY(4)	Qdbfspno	Offset from the start of the FDT header, Qdb_Qdbfh (page 56), to the next select/omit parameter for this select/omit statement.
4	4		BINARY(2)	Qdbfspln	Select/omit parameter length, 1 through 32,767.

Offset		Bit	Type	Field	Description
Dec	Hex				
6	6		CHAR(1)	Qdbfspin	Select/omit parameter attribute indicator. *X'00'* The parameter is a compare value. *X'01'* The parameter is a internal record format field. *X'02'* The parameter is an external record format field.
7	7		CHAR(1)	Qasopaf	Select/omit attribute flags.
7	7	0	BIT(1)	Qdbfsigc	Double-byte character set (DBCS) and/or graphic data indicator. If on, indicates the non-field compare value contains DBCS or graphic data.
7	7	1	BIT(1)	Qdbfshex	Hexadecimal data indicator. If on, indicates the non-field compare value is hexadecimal data.
7	7	2	BIT(1)	Qdbfsnul	Null value indicator. If on, indicates the non-field compare value is the null value.
7	7	3	BIT(5)	Reserved_57	Reserved.
8	8		BINARY(2)	Qdbfsppj	Select/omit parameter join reference number (JREF), 1 through » 256. « This field is not applicable if this file is not a join logical file or the compare value is a non-field value.
10	A		CHAR(10)	Reserved_58	Reserved.
20	14		CHAR(*)	Qdbfspvl	Select/omit parameter compare value or the record format field name. This is the compare value when Qdbfspin contains X'00'. This is the record format field name when Qdbfspin contains X'01' or X'02'.

Key Specification Array (Qdb_Qdbfk)

The key specification array (*Qdb_Qdbfk*) entries describe the record format fields used in defining the file access path.

Non-join logical files can have one key specification array for each file scope array entry.

Join logical files can have only one key specification array. The first scope array entry for the join logical file contains the offset to the file's key specification array.

You can locate this section with the offset (Qdbfksof (page 80)) in the scope array entry section, Qdb_Qdbfb.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		CHAR(10)	Qdbfkfld	Key statement field name. X'40's indicate the key statement is a *NONE key field.
10	A		CHAR(3)	Reserved_59	Reserved.
13	D		CHAR(1)	Qdbfksq	Key statement sequencing attribute flags.
13	D	0	BIT(1)	Qdbfksad	Ascending/descending sequence indicator. If on, indicates the descending sequence (*DESCEND).

Offset					
Dec	Hex	Bit	Type	Field	Description
13	D	1	BIT(2)	Qdbfksn	Numeric key field sequencing indicators. *B'00'* The numeric key field sequences as a string of unsigned binary data (UNSIGNED). *B'01'* The numeric key field ignores the sign of the field and sequences as absolute value data (ABSVAL). *B'10'* The numeric key field considers the sign of the field and sequences as signed value data (SIGNED).
13	D	3	BIT(1)	Reserved_60	Reserved.
13	D	4	BIT(1)	Qdbfksac	Alternate collating sequence indicator (ALTSEQ). If on, indicates the alternate collating sequence table applies to this key field.
13	D	5	BIT(1)	Qdbfkszf	Force zone sequencing indicator. If on, indicates the zone portion of the key field is zeroed so only the digit portion (furthest right four bits) is used in key sequencing (DIGIT). If off, the zone portion is not zeroed.
13	D	6	BIT(1)	Qdbfksdf	Force digit sequencing indicator. If on, indicates the digit portion of the key field is zeroed so only the zone portion (furthest left four bits) is used in key sequencing (ZONE). If off, the digit portion is not zeroed.
13	D	7	BIT(1)	Qdbfkft	Key statement external or internal name indicator. If on, indicates the field name is the external record format name.
14	E		CHAR(18)	Reserved_61	Reserved.

Distributed File Definition Section and Partition Key Array (Qdb_Qdbf_dis_pkeyarr)

The distributed file definition section and partition key array (*Qdb_Qdbf_dis_pkeyarr*) contains the node group name and library name for the distributed file and the record format fields used in defining the partition key for each scope entry.

You can locate this section with the offset Qdbfodis (page 80) in the scope array entry section, Qdb_Qdbfb.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		CHAR(10)	Qdbf_dis_ ndgpn	Distributed file node group name.
10	A		CHAR(10)	Qdbf_dis_ ndgpl	Distributed file node group library name.
20	14		BINARY(4)	Qdbf_dis_ nkyn	Number of partition key fields for this scope entry.
24	18		CHAR(40)	Reserved_ 121	Reserved.
64	40		ARRAY of CHAR(32)	Qdbf_dis_ pkeyarr	Distributed file partition key array.
64	40		CHAR(10)	Qdbf_dis_ kname	Partition key field name.
74	4A		CHAR(22)	Reserved_ 122	Reserved.

Journal Information (Qdb_Qdbfjoal)

The section *Qdb_Qdbfjoal* contains the journal information for the physical file. You can locate this section with offset Qdbfjorn (page 62) in the FDT header section, Qdb_Qdbfh (page 56).

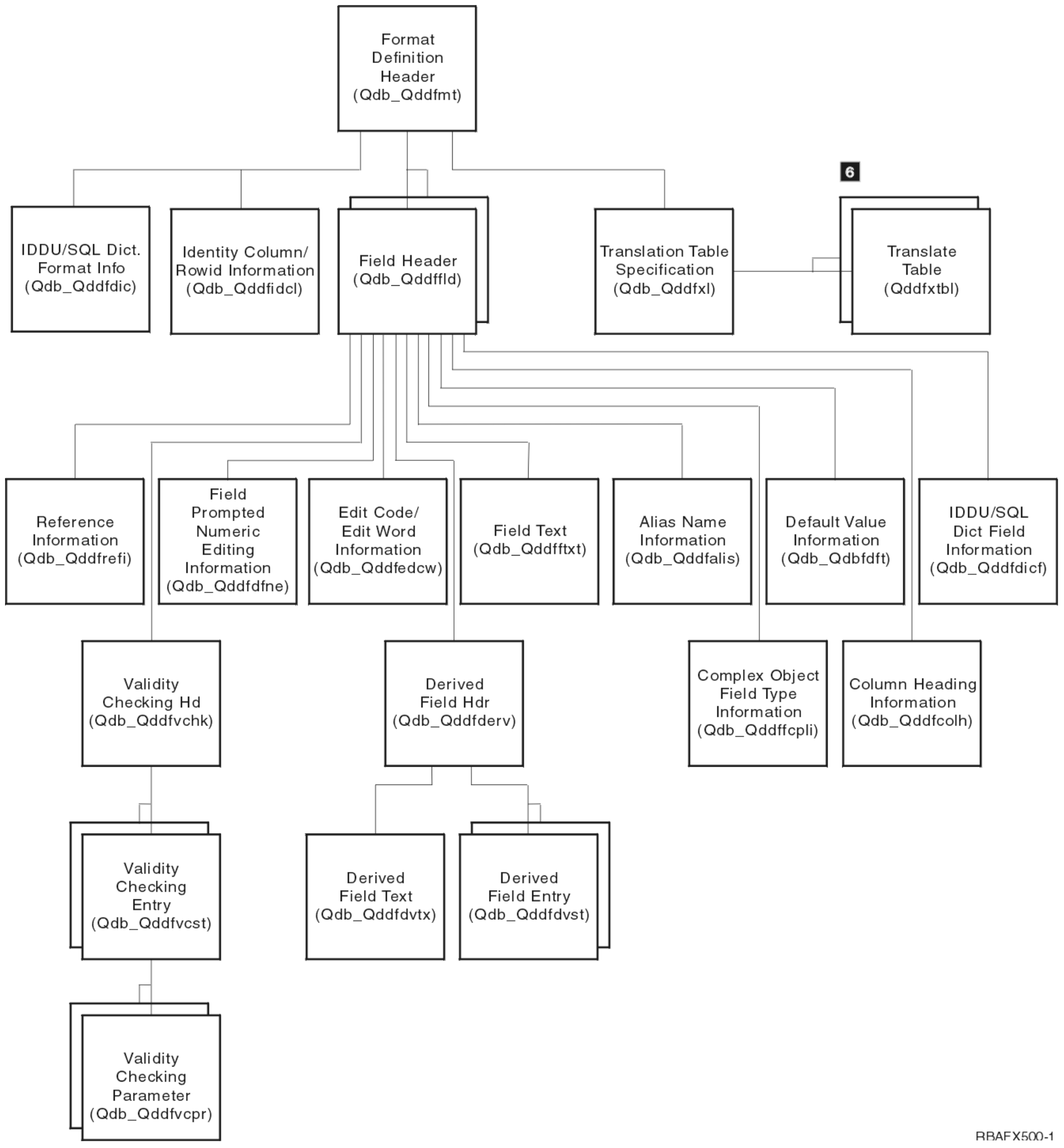
Offset					
Dec	**Hex**	**Bit**	**Type**	**Field**	**Description**
0	0		CHAR(10)	Qdbfojrn	Journal name.
10	A		CHAR(10)	Qdbfolib	Journal library name.
20	14		CHAR(1)	Qdbfojpt	Journaling options.
20	14	0	BIT(1)	Reserved_106	Reserved.
20	14	1	BIT(1)	Qdbfjbim	Before image indicator. If on, indicates the before images are being journaled.
20	14	2	BIT(1)	Qdbfjaim	After image indicator. If on, indicates the after images are being journaled.
20	14	3	BIT(1)	Reserved_107	Reserved.
20	14	4	BIT(1)	Qdbfjomt	Omit journal entries indicator. If on, indicates the open and close entries are being omitted from the journal.
20	14	5	BIT(3)	Reserved_108	Reserved.
21	15		CHAR(1)	Qdbfjact	Journaling options. *0* The file is not being journaled. *1* The file is being journaled.
22	17		CHAR(13)	Qdbfljrn	Last journaling date stamp. This is the date that corresponds to the most recent time that journaling was started. The date is in internal standard format (ISF), CYYMMDDHHMMSS.
35	23		CHAR(29)	Reserved_105	Reserved.

FILD0200 Format (Qdb_Qddfmt Structure)

FILD0200 provides the format used by the records of the specified file. This structure is also used by the QQQQRY API to get data from the named file. FILD0200 Format (page 84) shows how this information is organized. When more than one entry can appear, the figure indicates this as in **(6)**. Descriptions and offsets of the fields in this structure are in the tables immediately following FILD0200 Format (page 84).

The descriptions and offsets are available in the include source supplied on the system. You can see this source in source file H, member name QDBRTVFD, in the QSYSINC library.

FILD0200 Format



Format Definition Header (Qdb_Qddfmt)
6
IDDU/SQL Dict. Format Info (Qdb_Qddfdic)
Identity Column/ Rowid Information (Qdb_Qddfidcl)
Field Header (Qdb_Qddffld)
Translation Table Specification (Qdb_Qddfxl)
Translate Table (Qddfxtbl)
Reference Information (Qdb_Qddfrefi)
Field Prompted Numeric Editing Information (Qdb_Qddfdfne)
Edit Code/ Edit Word Information (Qdb_Qddfedcw)
Field Text (Qdb_Qddfftxt)
Alias Name Information (Qdb_Qddfalis)
Default Value Information (Qdb_Qdbfdft)
IDDU/SQL Dict Field Information (Qdb_Qddfdicf)
Validity Checking Hd (Qdb_Qddfvchk)
Derived Field Hdr (Qdb_Qddfderv)
Complex Object Field Type Information (Qdb_Qddffcpli)
Column Heading Information (Qdb_Qddfcolh)
Validity Checking Entry (Qdb_Qddfvcst)
Derived Field Text (Qdb_Qddfdvtx)
Derived Field Entry (Qdb_Qddfdvst)
Validity Checking Parameter (Qdb_Qddfvcpr)
RBAFX500-1

Format Definition Header (Qdb_Qddfmt)

The *Qdb_Qddfmt* section is always located at the beginning of the returned data area.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		BINARY(4)	Qddbyrtn	Bytes returned. The total length, in bytes, of the data returned.
4	4		BINARY(4)	Qddbyava	Bytes available. The total length, in bytes, of the format.
8	8		CHAR(24)	Reserved_62	Reserved.
32	20	0	CHAR(1)	Qddfmtf	Record format DBCS flags.
32	20	0	BIT(1)	Qddfrity	Double byte character set and/or graphic data. If on, indicates the format contains DBCS or graphic data.

Offset					
Dec	Hex	Bit	Type	Field	Description
32	20	1	BIT(1)	Qddfrilt	Double byte character set and/or graphic literals. If on, indicates the format contains DBCS or graphic literals.
32	20	2	BIT(1)	Qddfritx	Double byte character set record format text description. If on, indicates the text description contains DBCS data.
32	20	3	BIT(1)	Qddfrmep	Mapping error possible. If on, indicates the format contains fields that may return mapping errors.
32	20	4	BIT(1)	Qddfrdrv	Derived fields (logical files only). If on, indicates the format contains fields derived from fields in the physical file on which the logical file is based, or from fields in this logical file.
32	20	5	BIT(1)	Qddfrni	Neither or input-only files (logical files only). If on, indicates the format contains fields that cannot be used for input or output operations, or fields that can be used for input operations only.
32	20	6	BIT(1)	Qddfrdfi	Default values (physical files only). If on, indicates the format contains fields with default values (DFT).
32	20	7	BIT(1)	Qddfcato	Concatenated fields (logical files only). If on, indicates the format contains fields that are concatenations of two or more fields from the physical file.
33	21		BINARY(4)	Qddfxlto	Offset from the start of the Qdb_Qddfmt (page 85) header to the translate table specifications, Qddfxl (page 110).
37	25		BINARY(4)	Qddfrcao	Offset from the start of the Qdb_Qddfmt (page 85) header to the case selection specifications, Qddfcsl (page 110).
41	29		BINARY(4)	Qddfdico	Offset from the start of the Qdb_Qddfmt (page 85) header to the IDDU/SQL dictionary format information, Qddfdic (page 110) .
45	2D		BINARY(2)	Qddfrcid	Common coded character set identifier. Before using this field, see if Qddfrsid (page 87) is zero. If it is zero, not all character fields in the format use the same CCSID and this field is not valid.
47	2F		BINARY(2)	Qddfsrcd	Source file coded character set identifier. The CCSID for the character portion of the source file containing the DDS used to create the format.
49	31		BINARY(2)	Qddfrtcd	Format text coded character set identifier. The CCSID for the information about the text description.
51	33		BINARY(2)	Qddfrlcd	Long comment coded character set identifier. The CCSID for the information about the format content and purpose.
53	35		CHAR(7)	Reserved_64	Reserved.
60	3C		CHAR(1)	Qddftflgs	Format flags.
60	3C	0	BIT(1)	Qddfr12	Reserved.
60	3C	1	BIT(1)	Qddfucsd	If on, the format contains UCS-2 fields.
60	3C	2	BIT(1)	Qddfdlnk	If on, the format contains datalink fields.
60	3C	3	BIT(1)	Qddfdudt	If on, the format contains user-defined type fields.
60	3C	4	BIT(1)	Qddfdlob	If on, the format contains large object fields.
60	3C	5	BIT(1)	Qddfutfd	If on, the format contains a UTF-8 or UTF-16 field.
60	3C	6	BIT(2)	Reserved_114	Reserved.
61	3D		CHAR(1)	Qddflgs	Flags

Offset					
Dec	Hex	Bit	Type	Field	Description
61	3D	0	BIT(1)	Reserved_65	Reserved.
61	3D	1	BIT(1)	Qddfrvar	Variable length fields. If on, indicates the format contains variable length fields (VARLEN).
61	3D	2	BIT(1)	Qddfrgph	Graphic fields. If on, indicates the format contains graphic data fields.
61	3D	3	BIT(1)	Qddfrdtt	Date, time, or timestamp fields. If on, indicates the format contains data, time, or timestamp fields.
61	3D	4	BIT(1)	Qddfrnul	Null capable fields. If on, indicates the format contains null capable fields.
61	3D	5	BIT(1)	Qddfrsid	Common coded character set identifier flag. If on, indicates all character fields use the same CCSID.
61	3D	6	BIT(1)	Qddfesid	Explicit coded character set identifier flag. If on, indicates a CCSID was specified for the format file or for one or more fields in the format.
61	3D	7	BIT(1)	Reserved_66	Reserved.
62	3E		CHAR(4)	Reserved_67	Reserved.
66	42		BINARY(4)	Qddfrlen	Record length. The sum of the lengths of all format fields excluding neither fields.
70	46		CHAR(10)	Qddfname	Record format name.
80	50		CHAR(13)	Qddfseq	Level identifier. The modification level identifier of the format, used to verity the format has not changed since compile time, if LVLCHK(*YES) is requested.
93	5D		CHAR(50)	Qddftext	Text description (TEXT)
143	8F		BINARY(2)	Qddffldnum	Number of fields. The number of fields in the format. There is one field header for each field.
145	91		BINARY(4)	Qddf_Identity_ Off	Offset from the start of the Format header to the identity information, Qddfidcl (page 100)
256	100		Array of CHAR(*)	Qddffldx	Start of field definition array (Qdb_Qddffld).

Field Header (Qdb_Qddffld)

This section is located immediately after the Qdb_Qddfmt (page 85) header. The number of entries in this structure is defined by variable Qddffldnum (page 87) in the Qdb_Qddfmt header. This structure is to be defined at variable Qddffldx in the Qdb_Qddfmt header.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		BINARY(4)	Qddfdefl	Length of field header structure. The length of each occurrence of the field header structure, including all subsections.
4	4		CHAR(30)	Qddfldi	Internal field name. The name of the physical format field. If this is a logical format, the name of the physical field on which the logical field is based.
34	22		CHAR(30)	Qddflde	External field name. If this is a logical format, the logical format field name. If this is a physical format, the internal name is a duplicate of Qddflde.

Offset		Bit	Type	Field	Description
Dec	Hex				
64	40		CHAR(2)	Qddfftyp	Data type. *X'0000'* BINARY *X'0001'* FLOAT *X'0002'* ZONED DECIMAL *X'0003'* PACKED DECIMAL *X'0004'* CHARACTER *X'8004'* VAR CHARACTER *X'0005'* GRAPHIC *X'8005'* VAR GRAPHIC *X'0006'* DBCS-CAPABLE *X'8006'* VAR DBCS-CAPABLE *X'000B'* DATE *X'000C'* TIME *X'000D'* TIMESTAMP *X'4004'* BLOB/CLOB *X'4005'* DBCLOB *X'4006'* CLOB-OPEN *X'8044'* DATALINK-CHAR *X'8046'* DATALINK-OPEN *X'FFFF'* NULL
66	42		CHAR(1)	Qddffiob	Usage *X'01'* The field can be used for input only. *X'02'* Output only. *X'03'* Both input and output. *X'04'* Neither input nor output. *X'FF'* The usage is unknown.
67	43		BINARY(4)	Qddffobo	Output buffer offset. The offset of this field from the start of the output buffer.
71	47		BINARY(4)	Qddffibo	Input buffer offset. The offset of this field from the start of the input buffer.
75	4B		BINARY(2)	Qddffldb	Length. The length of the field. For character fields: the number of characters. For float fields: 4 for single, 8 for double. For variable length fields: the maximum the field can be plus 2. For date, time, or timestamp fields: the length of the formatted data. For graphic data fields: the number of bytes. For LOB fields: the number of bytes in the buffer.
77	4D		BINARY(2)	Qddffldd	Number of digits. The number of digits in the field. For numeric fields: the number of digits. For graphic data fields: the number of DBCS characters the field can contain.

Offset		Bit	Type	Field	Description
Dec	Hex				
79	4F		BINARY(2)	Qddffldp	Decimal positions. The number of position to the right of the decimal point.
81	51		CHAR(1)	Qddffkbs	Keyboard shift (RESHIFT) The keyboard shift attribute of the field. *X* Alphabetic only. *A* Alphameric shift. *N* Numeric shift. *S* Signed numeric. *Y* Numeric only. *D* Digits only. *M* Numeric only character. *W* Katakana. *H* Hexadecimal. *I* Inhibit keyboard entry. *J* DBCS only. *E* DBCS either. *O* DBCS open. »*B* Binary character.« *X'00'* No shift expected.
82	52		CHAR(1)	Qddffldst	Field status byte 1
82	52	0	BIT(1)	Qddffiat	Double-byte character set (DBCS) alternate type field. If on, indicates the alternate type for this field contains DBCS data.
82	52	1	BIT(1)	Qddffitx	Double-byte character set (DBCS) field text description. If on, indicates the text description contains DBCS data.
82	52	2	BIT(1)	Qddffich	Double-byte character set (DBCS) column headings. If on, indicates the column headings contains DBCS data.
82	52	3	BIT(1)	Qddffivc	Double-byte character set (DBCS) validity checking literals. If on, indicates the compare, range, or values literals contain DBCS data.
82	52	4	BIT(1)	Qddffrnd	Rounding. Rounding method for the field. If on, indicates round insignificant decimal digits. If off, indicates truncate insignificant decimal digits.
82	52	5	BIT(1)	Qddffcid	Character identifier flag. If on, indicates a character identifier was specified.
82	52	6	BIT(2)	Reserved_62	Reserved.
83	53		BINARY(2)	Qddfjref	Join reference (JREF) (logical files only). For fields whose names are specified in more than one physical file, this values identifies which physical file contains the field.
85	55		CHAR(1)	Qddffldst2	Field status byte 2.
85	55	0	BIT(1)	Qddffnul	Allow null value (ALWNULL). If on, indicates the null value is allowed for this field.
85	55	1	BIT(1)	Qddffdft	Column default value. If on, indicates the column does not have a default value.

Offset					
Dec	Hex	Bit	Type	Field	Description
85	55	2	BIT(1)	Qddffvar	If on, indicates the column is a variable length field.
85	55	5	BIT(5)	Reserved_70	Reserved.
86	56		CHAR(1)	Qddflgs2	Flags.
86	56	0	BIT(1)	Qddfcorr	Correlated field. If on, indicates this is a correlated field.
86	56	1	BIT(1)	Qddffrrn	File relative record number. If on, indicates this is a relative record number field.
86	56	2	BIT(5)	Reserved_71	Reserved.
86	56	7	BIT(1)	Qddffmep	Mapping errors possible. If on, indicates the field may return data mapping errors.
87	57		BINARY(2)	Qddfvarx	Variable field index. Index into the list of all variable field values for the query.
89	59		CHAR(2)	Reserved_72	Reserved.
91	5B		BINARY(2)	Qddflalc	Allocated length. The number of bytes allocated for the field in the fixed portion of the file. Or: Date/time/timestamp length. The number of bytes the based on field occupies.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Field**	**Description**
93	5D		CHAR(1)	Qddfdttf	Date format (DATFMT) or time format (TIMFMT), depending on the use of the field. This field is not valid unless Qddfftyp (page 88) is X′000B′, X′000C′, or X′000D′ except for the following cases. DATFMT and TIMFMT are valid on ′0002′X type logical file fields having based-on physical file fields that are ′000B′X and ′000C′X. DATFMT is valid on ′0003′X and ′0004′X type logical file fields having based-on physical file fields that are ′000B′X. Some DATFMTs are valid only for the the ′0002′X, ′0003′X, and ′0004′X fields having based-on physical file ′000B′X fields and are identified (by pseudo date) below. *X′FE′* The format associated with the job. *X′FF′* The format associated with the QDT. *X′01′* The *USA format. *X′03′* The *ISO format. *X′05′* The *EUR format. *X′07′* The *JIS format (date only). *X′09′* The SAA timestamp. *X′17′* The *MDY format (date only). *X′18′* The *DMY format (date only). *X′19′* The *YMD format (date only). *X′1A′* The *JUL format (date only). *X′1B′* The *HMS format (time only). *X′25′* The *CMDY format (pseudo date). *X′26′* The *CDMY format (pseudo date). *X′27′* The *CYMD format (pseudo date). *X′28′* The *MDYY format (pseudo date). *X′29′* The *DMYY format (pseudo date). *X′2A′* The *YYMD format (pseudo date). *X′2B′* The *YM format (pseudo date). *X′2C′* The *MY format (pseudo date). *X′2D′* The *YYM format (pseudo date). *X′2E′* The *MYY format (pseudo date). *X′30′* The *LONGJUL format (pseudo date).

Offset					
Dec	Hex	Bit	Type	Field	Description
94	5E		CHAR(1)	Qddfdtts	Date separator (DATSEP) or Time separator (TIMSEP) This field is not valid unless Qddfftyp (page 88) is X'000B', X'000C', or X'000D'. *X'00'* The separator associated with the job. *X'EE'* The implied separator is used. *'/'* The slash is used. *'-'* The dash is used. *'.'* The period is used. *' '* The blank is used. *':'* The colon is used.
95	5F		BINARY(2)	Qddfcsid	Common coded character set identifier (CCSID). *00000* The CCSID associated with the job is used. *65535* No data translation is done. *nnnnn* The CCSID.
97	61		BINARY(2)	Qddftsid	Text description common coded character set identifier. *00000* The CCSID associated with the job is used. *65535* No data translation is done. *nnnnn* The CCSID.
99	63		BINARY(2)	Qddfhsid	Column heading common coded character set identifier. *00000* The CCSID associated with the job is used. *65535* No data translation is done. *nnnnn* The CCSID.
101	65		BINARY(2)	Qddflsid	Long comment common coded character set identifier. *00000* The CCSID associated with the job is used. *65535* No data translation is done. *nnnnn* The CCSID.
103	67		CHAR(1)	Qddfldur	Labeled duration. The type of labeled duration this field defines. *X'00'* The field not a labeled duration. *X'0D'* Year/years. *X'0E'* Month/months. *X'0F'* Day/days. *X'10'* Hour/hours. *X'11'* Minute/minutes. *X'12'* Second/seconds. *X'13'* Microsecond/microseconds.

Offset		Bit	Type	Field	Description
Dec	Hex				
104	68		CHAR(1)	Qddfflgs	Flags. These flags indicate the user-specified value for the field at the time the format was created. These flags only apply to date, time, and timestamp fields. If the format was ever shared, these flags may not be applicable to the file for which the API was called.
104	68	0	BIT(1)	Qdddatfmt_dft	If on, indicates DATFMT was defaulted.
104	68	1	BIT(1)	Qdddatfmt_job	If on, indicates DATFMT(*JOB) was specified.
104	68	2	BIT(1)	Qdddatsep_dft	If on, indicates DATSEP was defaulted.
104	68	3	BIT(1)	Qdddatsep_job	If on, indicates DATSEP(*JOB) was specified.
104	68	4	BIT(1)	Qddtimfmt_dft	If on, indicates TIMFMT was defaulted.
104	68	5	BIT(1)	Qddtimsep_dft	If on, indicates TIMSEP was defaulted.
104	68	6	BIT(1)	Qddtimsep_job	If on, indicates TIMSEP(*JOB) was specified.
104	68	7	BIT(1)	QddSAAfmt	SAA format was specified.
105	69		BINARY(2)	Qddfwsid	Edit word common coded character set identifier. *00000* The CCSID associated with the job is used. *65535* No data translation is done. *nnnnn* The CCSID.
107	6B		CHAR(1)	Reserved_61	Reserved.
108	6C		CHAR(1)	Reserved_62	Reserved.
109	6D		BIN(2)	Reserved_63	Reserved.
111	6F		CHAR(1)	Qddflagco	Flags.
111	6F	0	BIT(3)	Reserved_64	Reserved.
111	6F	3	BIT(1)	Qddffucs	If on, indicates the column is a UCS-2 field.
111	6F	4	BIT(1)	Qddfudt	If on, indicates the column is a user-defined type field.
111	6F	5	BIT(1)	Qddf_Identity_ Col	If on, indicates the column is an identity column.
111	6F	6	BIT(1)	Qddf_Rowid_ Col	If on, indicates the column is a row ID column.
111	6F	7	BIT(1)	Qddfutf	If on, indicates the field is a UTF-8 or UTF-16 field.
112	70		CHAR(68)	Reserved_74	Reserved.
180	B4		BINARY(4)	Qddfcplx	Offset from the start of the field header to the field information if the field was a user-defined type, datalink, or large object. See structure Qdb_Qddfcpli.
184	B8		BINARY(4)	Qddfbmaxl	Maximum length of the large object field.
188	BC		BINARY(2)	Qddfbpadl	Pad length of the large object field.
190	BE		BINARY(4)	Qddfdicd	Offset from the start of the field header to the IDDU/SQL dictionary field information, Qddfdicf (page 110).
194	C2		BINARY(4)	Qddfdftd	Offset from the start of the field header to the default value description, Qddfdft (page 100).
198	C6		BINARY(4)	Qddfderd	Offset from the start of the field header to the derived field description (or to the concatenated field description if its file is externally described), Qddfderv (page 101).
202	CA		CHAR(6)	Reserved_75	Reserved.

Offset		Bit	Type	Field	Description
Dec	Hex				
208	D0		BINARY(4)	Qddftxtd	Offset from the start of the field header to the field text description, Qddftxt (page 99).
212	D4		CHAR(2)	Reserved_102	Reserved.
214	D6		BINARY(4)	Qddfrefd	Offset from the start of the field header to the field reference information, Qddfrefi (page 94).
218	DA		BINARY(2)	Qddfedtl	Length of the edit code/edit word for the field.
220	DC		BINARY(4)	Qddfedtd	Offset from the start of the field header to the edit code/edit word information, Qddfedcw (page 96).
224	E0		BINARY(2)	Reserved_76	Reserved.
226	E2		BINARY(4)	Qddfchd	Offset from the start of the field header to the column heading information, Qddfcolh (page 109).
230	E6		BINARY(2)	Qddfvckl	Length of validity checking data present for the field.
232	E8		BINARY(4)	Qddfvckd	Offset from the start of the field header to the validity checking data, Qddfvchk (page 97).
236	EC		BINARY(4)	Qddfxals	Offset from the start of the field header to the alias name entry.
240	F0		BINARY(4)	Qddffpnd	Offset from the start of the field header to the field prompted numeric editing information, Qddfdfne (page 95).
244	F4		CHAR(8)	Reserved_77	Reserved.
252	FC		CHAR(*)	Qddfvpx	Start of the variable portion of the field description.

Reference Information (Qdb_Qddfrefi)

You can locate the *Qdb_Qddfrefi* section with the offset Qddfrefd (page 94) in the field header section, Qdb_Qddffld.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		CHAR(1)	Qddfrcde	Modification flags.
0	0	0	BIT(1)	Qddfdupe	Modifications. If on, indicates the field has been modified.
0	0	1	BIT(1)	Qddfnmec	Name modification. If on, indicates the name of the field has been modified.
0	0	2	BIT(1)	Qddftypc	Data type modification. If on, indicates the data type of the field has been modified.
0	0	3	BIT(1)	Qddflenc	Field length modification. If on, indicates the length of the field has been modified.
0	0	4	BIT(1)	Qddfdecc	Precision modification. If on, indicates the precision of the field has been modified.
0	0	5	BIT(1)	Qddfedtc	Edit information modification. If on, indicates the edit information of the field has been modified.
0	0	6	BIT(1)	Qddfvc	Validity checking information modification. If on, indicates the validity checking information of the field has been modified.
0	0	7	BIT(1)	Qddfothr	Other modification. If on, indicates other information of the field has been modified.
1	1		CHAR(10)	Qddfrfil	Reference file name.

Offset					
Dec	Hex	Bit	Type	Field	Description
11	B		CHAR(10)	Qddfrlib	Reference file library.
21	15		CHAR(10)	Qddfrfmt	Referenced record format.
31	1F		CHAR(30)	Qddfrfld	Referenced field.
61	3D		CHAR(19)	Reserved_78	Reserved.

Field Prompted Numeric Editing Information (Qdb_Qddfdfne)

You can locate the *Qdb_Qddfdfne* section with the offset Qddffpnd (page 94) in the field header section, Qdb_Qddffld.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		CHAR(2)	Reserved_80	Reserved.
2	2		CHAR(1)	Qddfddts	Date separator (DATSEP) or Time separator (TIMSEP). *X'00'* This is not a date or time field. *1* The period (.). *2* The slash (/). *3* The colon (:). *4* The dash (-). *5* The comma (,).
3	3		CHAR(1)	Qddfddpc	Decimal point character. This field is not valid unless Qddfddts (page 95) contains X'00'. *1* The period (.). *2* The comma (,). *3* The colon (:). *4* The dollar ($). *5* No decimal point is used.
4	4		CHAR(1)	Qddfdtsc	Thousands separator character. This field is not valid unless Qddfddts (page 95) contains X'00'. *1* The period (.). *2* The comma (,). *3* The apostrophe ('). *4* The blank ( ). *5* No thousands separator is used.
5	5		CHAR(13)	Qnsi	Negative sign information.
5	5		CHAR(1)	Qddfdnsc	Display negative sign. This field is not valid unless Qddfddts (page 95) contains X'00'. *1* The negative sign is displayed for negative values. *2* The negative is not displayed for negative values.
6	6		CHAR(6)	Qddfdnsl	Left negative sign value. This field is not valid unless Qddfddts (page 95) contains X'00'.

Offset		Bit	Type	Field	Description
Dec	Hex				
12	C		CHAR(6)	Qddfdnsr	Right negative sign value. This field is not valid unless Qddfddts (page 95) contains X'00'.
18	12		CHAR(13)	Qcsi	Currency symbol information.
18	12		CHAR(1)	Qddfdcsv	Display currency symbol. This field is not valid unless Qddfddts (page 95) contains X'00'. *1* The currency symbol is displayed. *2* The currency symbol is not displayed.
19	13		CHAR(6)	Qddfdcsl	Left currency symbol value. This field is not valid unless Qddfddts (page 95) contains X'00'.
25	19		CHAR(6)	Qddfdcsr	Right currency symbol value. This field is not valid unless Qddfddts (page 95) contains X'00'.
31	1F		CHAR(1)	Qddfdpzv	Print zero value. This field is not valid unless Qddfddts (page 95) contains X'00'. *1* A zero value is displayed. *2* A zero value is not displayed.
32	20		CHAR(1)	Qddfdrlz	Replace leading zeros. This field is not valid unless Qddfddts (page 95) contains X'00'. *1* Leading zeros are replaced. *2* Leading zeros are not replaced.
33	21		CHAR(1)	Qddfdrlv	Leading zero replacement value. This field is not valid unless Qddfddts (page 95) contains X'00'. *1* Blanks ( ). *2* Asterisks (*). *3* Blanks ( ) and the left currency symbol is shifted right.
34	22		CHAR(1)	Qddfdlzo	Single leading zero. This field is not valid unless Qddfddts (page 95) contains X'00'. *1* A zero is displayed to the left of the decimal point when there are no significant digits to the left of the decimal. *2* A zero is not displayed to the left of the decimal point.
35	23		CHAR(29)	Reserved_81	Reserved.

Edit Code/Edit Word Information (Qdb_Qddfedcw)

You can locate the *Qdb_Qddfedcw* section with the offset Qddfedtd (page 94) in the field header section, Qdb_Qddffld.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		CHAR(2)	Qddfecdi	Edit code information.
0	0		CHAR(1)	Qddfecde	Edit code (EDTCDE). Edit code for the field when it is referred to during display or print file creation.

Offset		Bit	Type	Field	Description
Dec	Hex				
1	1		CHAR(1)	Qddfecdx	Floating currency symbol. * Asterisk protection: asterisks are displayed to the left of significant digits. A currency symbol indicates the symbol displayed to the left the significant digits.
2	2		CHAR(14)	Reserved_79	Reserved
16	10		CHAR(*)	Qddfewd	Edit word (EDTWRD). The form in which the field values are displayed.

Validity Checking Information (Qdb_Qddfvchk)

You can locate the *Qdb_Qddfvchk* section with the offset Qddfvckd (page 94) in the field header section, Qdb_Qddffld.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		BINARY(2)	Qddfvcnume	Number of validity check entries.
2	2		CHAR(14)	Reserved_82	Reserved.
16	10		CHAR(*)	Qddfvcen	Validity checking entry array.

Validity Checking Entry (Qdb_Qddfvcst)

The first validity checking entry starts at Qddfvcen in the validity checking information section, Qdb_Qddfvchk (page 97).

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		CHAR(1)	Qddfvccd	DDSI keyword identifier.
					X'63' CHKMSGID
					X'64' CHECK(ME)
					X'66' CHECK(FE)
					X'67' CHECK(MF)
					X'71' RANGE
					X'72' VALUES
					X'73' COMP(GT)
					X'74' COMP(GE)
					X'75' COMP(EQ)
					X'76' COMP(NE)
					X'77' COMP(LE)
					X'78' COMP(LT)
					X'79' COMP(NL)
					X'7A' COMP(NG)
					X'A0' CHECK(M10)
					X'A1' CHECK(M11)
					X'A2' CHECK(VN)
					X'A3' CHECK(AB)
					X'A5' CHECK(VNE)
					X'A6' CHECK(M10F)
					X'A7' CHECK(M11F)
1	1		BINARY(2)	Qddfvcnump	Number of parameters.
3	3		BINARY(2)	Qddfvcel	Length of this validity checking entry.
5	5		CHAR(11)	Reserved_83	Reserved.
16	10		CHAR(*)	Qddfvcpm	Validity checking parameter array.

Validity Checking Parameter (Qdb_Qddfvcpr)

The first validity checking parameter starts at Qddfvcpm (page 98) in the validity checking entry section, Qdb_Qddfvcst.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		BINARY(2)	Qddfvcpl	Length of validity checking parameter Qddfvcpv.
2	2		CHAR(14)	Reserved_84	Reserved.
16	10		CHAR(*)	Qddfvcpv	Validity checking parameter value.

Complex Object Field Type Information (Qdb_Qddfcpli)

You can locate the *Qdb_Qddfcpli* section with the offset Qddfcplx (page 93) in the field header section, Qdb_Qddffld.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		BINARY(4)	Qddflenu	Length of the user-defined type name.
4	4		CHAR(128)	Qddfnudt	User-defined type name.
132	84		CHAR(10)	Qddfludt	User-defined type library name.
142	8E		CHAR(1)	Qddfdlink	Link control. *N* No link control. *F* File link control.
143	8F		CHAR(1)	Qddfdinte	Link integrity. Linked files are under control of the database if the field is a datalink. *A* All under control. *S* Selective control. This value is not supported yet.
144	90		CHAR(2)	Qddfdrper	Read permission. The file system controls authority to read a file if the field is a datalink. *FS* File system. *DB* Database.
146	92		CHAR(2)	Qddfdwper	Write permission. The file system controls authority to write to a file if the field is a datalink. *FS* File system. *BL* Blocked.
148	94		CHAR(1)	Qddfdreco	Recovery. The database manager will recover the file if the field is a datalink. *Y* Yes. This value is not supported yet. *N* No.
149	95		CHAR(1)	Qddfdunlk	On unlink. The database manager will either restore the file owner on an unlink, or delete the file when unlinking the file. *R* Restore the owner. *D* Delete the file.
150	96		CHAR(10)	Reserved_150	Reserved.

Field Text (Qdb_Qddfftxt)

You can locate the *Qdb_Qddfftxt* section with the offset Qddftxtd (page 94) in the field header section, Qdb_Qddffld.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		CHAR(50)	Qddfftxt	Text (TEXT). Text description of the field.

Alias Name Structure (Qdb_Qddfalis)

You can locate this section with the offset Qddfxals located in the field header section, Qdb_Qddffld.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		BINARY(2)	Qddfalsl	Length of alternative name Qddfalsn (page 100).
2	2		CHAR(14)	Reserved_85	Reserved.
16	10		CHAR(258)	Qddfalsn	Alternative name (ALIAS).

Default Value Description Information (Qdb_Qddfdft)

You can locate the *Qdb_Qddfdft* section with the offset Qddfdftd (page 93) in the field header section, Qdb_Qddffld.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		BINARY(2)	Qddfdftl	Length of default information.
2	2		CHAR(1)	Qddfdfta	Default attributes.
2	2	0	BIT(1)	Qddfdfig	DBCS or graphic default. If on, indicates the default is a DBCS or graphic literal.
2	2	1	BIT(1)	Qddfdfhx	Hex default. If on, indicates the default is a hexadecimal literal.
2	2	2	BIT(1)	Qddfndft	Null default. If on, indicates the default is null.
2	2	3	BIT(2)	Reserved_86	Reserved.
2	2	5	BIT(1)	Qddfdcur	DATE, TIME, or TIMESTAMP default. On indicates the default is CURRENT_DATE, CURRENT_TIME, or CURRENT_TIMESTAMP.
2	2	6	BIT(1)	Reserved_109	Reserved.
2	2	7	BIT(1)	Qddfdftk	DFT or DFTVAL keyword. If on, indicates the DFTVAL keyword was specified.
3	3		CHAR(13)	Reserved_87	Reserved.
16	10		CHAR(*)	Qddfdftv	Default (DFT) or (DFTVAL). A value of USER indicates that the default value for this field is the job's current user.

Identity Column Information (Qdb_Qddfidcl)

You can locate the *Qdb_Qddfidcl* section with the offset Qddf_Identity_Off in the field header section, Qdb_Qddffld.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		BINARY(2)	Qddf_Id_Len	Length of IDENTITY information.
2	2		DECIMAL(31,0)	Qddf_Id_Orig_ Start_With	Original START WITH value.
18	12		DECIMAL(31,0)	Qddf_Id_Curr_ Start_With	Current START WITH value.
34	22		BINARY(4)	Qddf_Id_ Increment_By	INCREMENT BY value.
38	26		DECIMAL(31,0)	Qddf_Id_ Minimum	MINIMUM value.

Offset		Bit	Type	Field	Description
Dec	Hex				
54	36		DECIMAL(31,0)	Qddf_Id_ Maximum	MAXIMUM value.
70	46		CHAR(1)	Qddf_Id_Cycle	CYCLE indicator. 1 = Cycling will occur. 0 = Cycling will not occur.
71	47		CHAR(1)	Qddf_Id_Order	ORDER mode indicator. 1 = Values are generated in order of request. 0 = Values do not need to be generated in order of request.
72	48		CHAR(1)	Qddf_Id_ Generate	Identity GENERATE indicator. » 1 = GENERATE BY DEFAULT. 0 = GENERATE ALWAYS. «
73	49		BINARY(4) UNSIGNED	Qddf_Id_Curr_ Cache	CACHE value.
77	4D		CHAR(1)	Qddf_Rowid_ Generate	Rowid GENERATE indicator. » 1 = GENERATE BY DEFAULT. 0 = GENERATE ALWAYS. «
78	4E		CHAR(53)	Qddf_Id_ Reserved1	Reserved.

Derived Field Description Information

The derived field structure is a stack of operators and operands in postfix notation. **Postfix notation** is a method of forming mathematical expressions in which each operator is preceded by its operands and indicates the operation to be performed on the operands or the intermediate results that precede it. For example:

```
A + B
would be:
A B +
```

Numeric operands and character operands cannot be mixed in one derived field description. If numeric operands are specified, the resulting field attributes must be numeric. If character operands are specified, the resulting field attributes must be character or DBCS. Character and DBCS only fields cannot be mixed in one derived field description.

Substringing DBCS fields is allowed, although the data is treated as character data, that is, there is no true double-byte substring support. This applies to query formats only.

Derived Field Header (Qdb_Qddfderv)

You can locate this section, *Qdb_Qddfderv*, with the offset Qddfderd (page 93) in the field header section, Qdb_Qddffld.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		BINARY(4)	Qddfdvtl	Length of derived field information Qddfderv (page 101).
4	4		BINARY(2)	Qddfdvnume	Number of derived field entries. 0 indicates it is a concatentated field.
6	6		BINARY(4)	Qddfdvot	Offset from the start of this header to the derived field text (or to the concatenated field text), Qddfdvtx (page 109) .
10	A		CHAR(6)	*	Reserved.

Offset		Bit	Type	Field	Description
Dec	Hex				
16	10		CHAR(*)	Qddfdven	Derived field entry.

Derived Field Entry (Qdb_Qddfdvst)

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		BINARY(4)	Qddfdvln	Length of derived field entry, Qddfdvst.
4	4		BINARY(2)	Qddfdtyp	Derived field entry type. *0* A field operand. *1* A constant operand. *2* An operator.
6	6		CHAR(*)	Qddfdv	The union of the Field operand (Qdddvof), Constant operand (Qddffvoc), and Operator entry (Qddfdvo).

Field Operand Entry (Qdb_Qddfdvof)

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		CHAR(30)	Qddfdvon	Field name.
30	1E		BINARY(2)	Qddfdvjr	Join reference (JREF). Relative file number of the physical file containing the external file referenced. *0* The fields previously defined in this format are searched for the field name. If the field is not found, the based on file formats are searched. If the field name is found in more than one file format, an error is signalled. *n* The file containing the field name.
32	20		BINARY(2)	Qddfdv01	Starting position. The starting position in the field of the substring (SST) specified.
34	22		BINARY(2)	Qddfdvo2	Ending position. The ending position in the field of the substring (SST) specified.
36	24		BINARY(2)	Qddfqdtnum	Qdt from which this correlated field originates (only applicable for SQL subqueries.
38	26		CHAR(20)	*	Reserved.

Constant Operand Entry (Qdb_Qddfdvoc)

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		*	Qddfdvoh	Constant operand header.
0	0		BINARY(4)	Qddfdvol	Length of constant Qddfdvov (page 104).
4	4		CHAR(1)	Qca	Constant attributes.

Offset		Bit	Type	Field	Description
Dec	Hex				
4	4	0	BIT(1)	Qddfdvci	DBCS constant. If on, indicates the constant is a DBCS-open literal.
4	4	1	BIT(1)	Reserved_90	Reserved.
4	4	2	BIT(1)	Qddfdvcc	Character constant type. If on, indicates the constant is an unquoted character string not bracketed by single quotes. Imbedded quotes are represented with a single quote. If off, indicates it is quoted, bracketed by single quotes. Imbedded quotes are represented with two single quotes.
4	4	3	BIT(1)	Qddfdvac	Assume character constant. If on, indicates the system assumes this is a character constant.
4	4	4	BIT(1)	Qddfdvco	DBCS-only literal. If on, indicates the constant is a DBCS-only literal. This attribute is not valid if the DBCS constant attribute, Qddfdvci, is off.
4	4	5	BIT(1)	Qddfdvsr	Special register. If on, indicates this constant is a special register defined by Qddfdvrc.
4	4	6	BIT(1)	Qddfdvnl	Null indicator. If on, indicates the constant is a null literal.
4	4	7	BIT(1)	Reserved_91	Reserved.
5	5		CHAR(1)	Qddfdvrc	Special register constant. Defined by special register constants, can only be specified if Qddfdvsr is on.
6	6		CHAR(1)	Qddfdvft	Date constant format (DATFMT) or Time constant format (TIMFMT) *X'FE'* Format associated with the job is used. *X'FF'* Format associated with QDT is used. *X'01'* The *USA format. *X'03'* The *ISO format. *X'05'* The *EUR format. *X'07'* The *JIS format. *X'09'* The SAA timestamp format. *X'17'* The *MDY date format. *X'18'* The *DMY date format. *X'19'* The *YMD date format. *X'1A'* The *JUL date format. *X'1B'* The *HMS time format.

Offset					
Dec	Hex	Bit	Type	Field	Description
7	7		CHAR(1)	Qddfdvsp	Date constant separator (DATSEP) or Time constant separator (TIMSEP) *X'00'* Default separator associated with job is used. *X'EE'* The implied separator is used. *'/'* The slash. *'-'* The dash. *'.'* The period. *','* The comma. *' '* The blank. *':'* The colon.
8	8		CHAR(2)	Reserved_92	Reserved.
10	A		BINARY(2)	Qddfdvcd	Constant coded character set identifier (CCSID).
13	C		CHAR(1)	Qddfcflg	Constant flags.
13	C	0	BIT(2)	Reserved_93	Reserved.
13	C	2	BIT(1)	Qddfglit	Graphics literal. If on, indicates this is a graphics literal.
13	C	3	BIT(5)	Reserved_94	Reserved.
14	E		CHAR(29)	Reserved_95	Reserved.
43	2B		CHAR(*)	Qddfdvov	Derived constant. The external form of the constant.

Operator Entry (Qdb_Qddfdvo)

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		CHAR(1)	Qddfdvop	Derived operator. **Operators requiring three operands:** *X'27'* Substring

Offset		Bit	Type	Field	Description
Dec	Hex				
					Operators requiring two operands: *X'01'* Concatenation *X'04'* Addition *X'05'* Subtraction *X'06'* Multiplication *X'07'* Division *X'08'* Minimum *X'09'* Maximum *X'1A'* X to the Y power *X'1B'* Binary OR *X'1C'* Binary XOR *X'1D'* Binary AND *X'24'* Strip leading *X'25'* Strip tailing *X'26'* Strip both *X'35'* Compute *X'41'* String position *X'80'* Remainder

Offset		Bit	Type	Field	Description
Dec	Hex				
					Operators requiring one operand: *X'02'* Map *X'03'* Direct map *X'0A'* Absolute value *X'0B'* Translate *X'0C'* Natural logarithm *X'0D'* Exponential *X'0E'* Sine *X'0F'* Cosine *X'10'* Tangent *X'11'* Cotangent *X'12'* Arc sine *X'13'* Arc cosine *X'14'* Arc tangent *X'15'* Hyperbolic sine *X'16'* Hyperbolic cosine *X'17'* Hyperbolic tangent *X'18'* Hyperbolic arctangent *X'19'* Square root *X'1E'* Binary NOT *X'1F'* Negation *X'23'* Length *X'29'* Year *X'2A'* Month *X'2B'* Day *X'2C'* Days. *X'2D'* Hour *X'2E'* Minute *X'2F'* Second *X'30'* Microsecond *X'31'* Date *X'32'* Time *X'34'* Hex

Offset		Bit	Type	Field	Description
Dec	Hex				
					Operators requiring one operand continued: *X'36'* Test translate CCSID *X'37'* Translate monocase *X'3C'* Node number *X'3D'* Cast *X'47'* Partition *X'48'* Node name *X'83'* Log (base 10) *X'84'* Anti log (base 10) *X'85'* Digits *X'86'* Char *X'8F'* Graphic representation of character *X'90'* Character representation of graphic
					Label duration operators: *X'87'* Year *X'88'* Month *X'89'* Day *X'8A'* Hour *X'8B'* Minute *X'8C'* Second *X'8D'* Microsecond
					Operators requiring one to many operands: *X'3A'* Hash function
					Operators requiring two to many operands: *X'28'* Null values *X'3E'* Case Expression
					Operators requiring one or two operands: *X'33'* Timestamp
					Group by operators: All require one operand except count that requires one or two. *X'A1'* Count *X'A3'* Sum *X'A4'* Minimum *X'A5'* Maximum *X'B0'* Average *X'B1'* Standard deviation *X'B2'* Variance

Offset		Bit	Type	Field	Description
Dec	Hex				
1	1		CHAR(2)	Qddfdvxnum	Translate table index or case selection specification index. This field is valid only if Qddfdvop (page 104) is X'OB' or X'3E'.
1	1		CHAR(1)	Qddfdvdtfmt	Operator date format index.
1	1		CHAR(1)	Qddfdvdtsep	Operator date separator index.
3	3		CHAR(1)	Qddfdvfm	Operator date format (DATFMT) or Operator time format (TIMFMT). *X'FE'* Format associated with the job is used. *X'FF'* Format associated with QDT is used. *X'01'* The *USA format. *X'03'* The *ISO format. *X'05'* The *EUR format. *X'07'* The *JIS format. *X'09'* The SAA timestamp format. *X'17'* The *MDY date format. *X'18'* The *DMY date format. *X'19'* The *YMD date format. *X'1A'* The *JUL date format. *X'1B'* The *HMS time format.
4	4		CHAR(1)	Qddfdvsa	Operator date separator (DATSEP) or Operator time separator (TIMSEP) *X'00'* Default separator associated with job is used. *X'EE'* The implied separator is used. *'/'* The slash. *'-'* The dash. *'.'* The period. *','* The comma. *' '* The blank. *':'* The colon.
5	5		BINARY(2)	Qddfdvno	Number of operands.
7	7		CHAR(1)	Qoa	Operator attributes.
7	7	0	BIT(1)	Reserved_96	Reserved.
7	7	1	BIT(1)	Qddfdvdttm	Operator date format and separator source. If on, indicates Qddfdvdtfmt and Qddfdvdtsep are used as the date format and separator with the CHAR operator. Qddfdvfm and Qddfdvsa are used as the time format and separator with the CHAR operator.
7	7	2	BIT(1)	Reserved_n	Reserved.
7	7	3	BIT(1)	Qddfdvdf	Group operators. If on, do not include duplicate field values in group by operation. If off, include duplicate field values in group by operation.
7	7	4	BIT(1)	Reserved_97	Reserved.

Offset					
Dec	Hex	Bit	Type	Field	Description
7	7	5	BIT(1)	Qddffunc_char	Number of characters option. If on, the result of the operator is based on the number of characters. If off, the result of the operator is based on the number of bytes. This field is only applicable when Qddfdvop is POSSTR(X’41’), LENGTH(X’23’), or SUBSTRING(X’27).
7	7	6	BIT(2)	Reserved_115	Reserved.
8	8		CHAR(2)	Reserved_98	Reserved.
10	A		CHAR(1)	Qddfd_ decptchar	The character to use for the decimal point. Only applicable if Qddfdvop is a CAST(X’3D’) and one of the operands is numeric and the other is character, or if Qddfdvop is a CHAR(X’86’) and the first operand is packed decimal.
11	B		BIN(4)	Qddfdo_func_ def	Offset from the beginning of this derived field entry (Qdb_Qddfdvst) to the Function Name Specification section, Qddfunc_def (page 111) . If this offset is specified, then the function is resolved to using the name in the Function Name Specification section. If the Function Name section is specified, all entries in this operator section are ignored except for the number of operands for the function, Qddfdvno, which is required to be set, and the duplicate field values indicator, Qddfdvdf, which can be optionally set. decimal.
15	F		CHAR(11)	Reserved_101	Reserved.

Derived Field Text Information (Qdb_Qddfdvtx)

You can locate the *Qdb_Qddfdvtx* section with the offset Qddfdvot (page 101) in the Derived Field Header section, Qdb_Qddfderv.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		BINARY(2)	Qddfdvlt	Length of derived field text information or concatenated field text information.
2	2		CHAR(*)	Qddfdtxt	Derived field text description or concatenated field text description.

Column Heading Information (Qdb_Qddfcolh)

You can locate the *Qdb_Qddfcolh* section with the offset Qddfchd (page 94) in the field header section, Qdb_Qddffld.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		CHAR(20)	Qddfch1	Column heading #1. The first column heading specified on the COLHDG DDS keyword without the quotes.
20	14		CHAR(20)	Qddfch2	Column heading #2. The second column heading specified on the COLHDG DDS keyword without the quotes.
40	28		CHAR(20)	Qddfch3	Column heading #3. The third column heading specified on the COLHDG DDS keyword without the quotes.

IDDU/SQL Dictionary Format Information (Qdb_Qddfdic)

You can locate the *Qdb_Qddfdic* section with the offset Qddfdico (page 86) in the field header section, Qdb_Qddffld.

Offset		Bit	Type	Field	Description
Dec	**Hex**				
0	0		CHAR(32)	Reserved_100	Reserved.
32	20			Qddfdicm	Format definition long comment information.
32	20		BINARY(2)	Qddfdilt	Length of format definition long comment information, Qddfdicm.
34	22		CHAR(*)	Qddfditx	Format definition long comment.

IDDU/SQL Dictionary Field Information (Qdb_Qddfdicf)

You can locate the *Qdb_Qddfdicf* section with the offset Qddfdicd located in the field header section, Qdb_Qddffld.

Offset		Bit	Type	Field	Description
Dec	**Hex**				
0	0		CHAR(16)	*	Reserved.
16	10			Qddfdfco	Field definition long comment.
16	10		BINARY(2)	Qddffcl	Length of field definition long comment Qddfdfco.
18	12		CHAR(*)	Qddfdfct	Field definition comment text.

Translate Table Specification (Qdb_Qddfxl)

You can locate the *Qdb_Qddfxl* section with the offset Qddfxlto (page 86) in the field header section, Qdb_Qddffld.

Offset		Bit	Type	Field	Description
Dec	**Hex**				
0	0		BINARY(2)	Qddfxlnum	Number of elements in the translate table array.
2	2			Qddfxarr	Translate table array.
2	2		CHAR(10)	Qddfxtnm	Translate table name.
12	C		CHAR(10)	Qddfxtln	Translate table library name.
22	16		BINARY(2)	Qddfxcid	Translate table constant coded character set identifier.
24	18		CHAR(10)	Reserved_99	Reserved.
34	22		CHAR(256)	Qddfxtbl	Translate table.

Case Selection Specification (Qdb_Qddfcsl)

You can locate the *Qdb_Qddfcsl* section with the offset Qddfrcao in the field header section, Qdb_Qddffld. For a description of selection specifications, see QDBQS in the "Query (QQQQRY) API" on page 328 API.

Offset		Bit	Type	Field	Description
Dec	**Hex**				
0	0		BINARY(2)	Qddfcsnum	Number of elements in the case selection specification array.

Offset		Bit	Type	Field	Description
Dec	Hex				
2	2		BINARY(4)	Qddfcln	Length of this plus the length of all the selection specifications.
6	6		CHAR(10)	Reserved	Reserved.
16	10		Array of BINARY(4)	Qddfcao	Offset to the selection specification. Offset is from the start of Qdb_qdffcsl.

Function Name Specification (Qdb_Qddfunc_def)

You can locate the *Qdb_Qddfunc_def* section with the offset Qddfdo_func_def (page 109) in the derived operator entry section, Qdb_Qddfdvo.

This section can only be specified when used in conjunction with the "Query (QQQQRY) API" on page 328 API.

This section can be used to reference a function by name rather than opcode qddfdvop. It can be used to resolve to existing built-in functions provided by the database or to user-defined functions defined in the SYSROUTINE SQL catalog in the QSYS2 library. Resolution is based on function name, number of parameters, compatible parameters and library list, in that order.

See the DB2 UDB for iSeries SQL Reference book for more information on user-defined functions and the SYSROUTINE catalog.

Offset		Bit	Type	Field	Description
Dec	Hex				
0	0		CHAR(20)	Reserved	Reserved.
20	14		CHAR(10)	Qddfunc_libname	Library where function can be found. Special values follow. *' '* Blank. Use the path (library list) to find the function. *'QSYS2'* Use the built-in operator provided by the database.
30	1E		BIN(2)	Qddfunc_namelen	Length of function name in Qddfunc_funcname.
32	20		CHAR(128)	Qddfunc_funcname	Name of function to resolve.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Field**	**Description**
					Built-in functions provided by the database in library QSYS2: '+' Addition. Two operands '-' Subtraction. Two operands '*' Multiplication. Two operands '/' Division. Two operands *'ABS'* Absolute value. One operand *'ACOS'* Arc cosine. One operand *'ANTILOG'* Antilog. One operand *'ASIN'* Arc sine. One operand *'ATAN'* Arc tangent. One operand *'ATANH'* Hyperbolic arc tangent. One operand *'AVG'* Average. One operand *'CHAR'* Character. One to two operands *'COALESCE'* First non-null value. Two to N operands *'CONCAT'* Concatenation. Two operands *'COS'* Cosine. One operand *'COSH'* Hyperbolic cosine. One operand *'COT'* Co-tangent. One operand *'COUNT'* Count. One operand *'CURDATE'* Current date. Zero operand *'CURTIME'* Current time. Zero operand *'DATE'* Date. One operand *'DAY'* Day. One operand *'DAYOFMONTH'* Day of month. One operand *'DAYOFWEEK'* Day of week. One operand *'DAYOFYEAR'* Day of year. One operand

Offset					
Dec	Hex	Bit	Type	Field	Description
					Built-in functions *(continued)* *'DAYS'* Days. One operand *'DECIMAL'* Decimal of operand. One operand *'DEGREES'* Degrees. One operand *'DIGITS'* Character form of number. One operand *'DOUBLE'* Double precision. One operand *'EXP'* Natural log to the power. One operand *'FLOAT'* Floating point. One operand *'FLOOR'* Integer. One operand *'HASH'* Hash value. One to N operands *'HEX'* Hex value. One operand *'HOUR'* Hour. One operand *'IFNULL'* First non-null value. Two operands *'INT'* Integer. One operand *'LAND'* Logical AND. Two operands *'LCASE'* Lower case. One operand *'LEFT'* Left N characters. Two operands *'LENGTH'* Length. One operand *'LN'* Natural log. One operand *'LNOT'* Logical NOT. One operand *'LOCATE'* Search string in source string. Two to three operands *'LOG'* Base 10 log. One operand *'LOR'* Logical OR. Two operands *'LOWER'* Lower case. One operand *'LTRIM'* Remove leading blanks. One operand

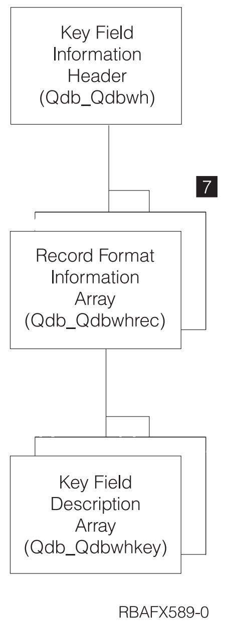
Offset		Bit	Type	Field	Description
Dec	Hex				
					Built-in functions *(continued)* *'MAX'* Max. One operand *'MAX'* Max. Two to N operands *'MICROSECOND'* Microsecond. One operand *'MIN'* Min. One operand *'MIN'* Min. Two to N operands *'MINUTE'* Minute. One operand *'MOD'* Modulo. Two operands *'MONTH'* Month. One operand *'NOW'* Current timestamp. Zero operands *'POSSTR'* Search string in source string. Two operands *'POWER'* Raise to power of. Two operands *'QUARTER'* Quarter. One operand *'REAL'* Single precision float. One operand *'RTRIM'* Trim trailing blanks. One operand *'SECOND'* Second. One operand *'SIN'* Sine. One operand *'SINH'* Hyperbolic sine. One operand *'SMALLINT'* Small integer. One operand *'SQRT'* Square root. One operand *'STDDEV'* Standard deviation. One operand *'SUBSTR'* Substr. Two to three operands *'SUM'* Sum. One operand

Offset					
Dec	**Hex**	**Bit**	**Type**	**Field**	**Description**
					Built-in functions *(continued)* *'TAN'* Tangent. One operand *'TANH'* Hyperbolic tangent. One operand *'TIME'* Time. One operand *'TIMESTAMP'* Timestamp. One to two operands *'TRANSLATE'* Translate. One to four operands *'UCASE'* Uppercase. One operand *'UPPER'* Uppercase. One operand *'VALUE'* First non-null value. Two to N operands *'VARCHAR'* Varchar. One to three operands *'VARGRAPHIC'* Vargraph. One to three operands *'VARIANCE'* Variance. One operand *'WEEK'* Week. One operand *'XOR'* Logical exclusive OR. Two operands *'YEAR'* Year. One operand *'ZONED'* Zoned. One to four operands

FILD0300 Format (Key Field Information)

FILD0300 provides detailed information for key fields of each record format of the specified file. This structure is used by the QQQQRY API to get data from the named file. FILD0300 Format (page 115) shows how this information is organized. When more than one entry can appear, the figure indicates this as in **(7)**. To get a description of all the fields contained in this structure and to determine the offsets, see the include source supplied on the system. An offset to the key field information array of each record format is provided in the record format information structure. If 0 is returned for this offset, this record format has no key field. If -1 is returned for this offset, the size of the receiver provided is insufficient to hold the returned data. You can see this source in source file H, member name QDBRTVFD, in the QSYSINC library.

FILD0300 Format


Key Field Information Header (Qdb_Qdbwh)
7
Record Format Information Array (Qdb_Qdbwhrec)
Key Field Description Array (Qdb_Qdbwhkey)
RBAFX589-0

Key Information Header (Qdb_Qdbwh)

The *Qdb_Qdbwh* section is always located at the beginning of the returned data area.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		BINARY(4)	Byte_Ret	Bytes returned. The total length, in bytes, of the data returned.
4	4		BINARY(4)	Byte_Avail	Bytes available. The total length, in bytes, of the key information.
8	8		BINARY(2)	Max_Key_Len	Maximum key length. The maximum length, in bytes, of any of the keys.
10	A		BINARY(2)	Key_Count	File generic key field count.
12	C		CHAR(10)	Reserved	Reserved.
22	16		BINARY(2)	Fmt_Counts	Number of formats for the file.

Record Format Key Information Array (Qdb_Qdbwhrec)

The *Qdb_Qdbwhrec* section is located immediately after the Qdb_Qdbwh (page 116) header. This is a linked list. There is a format record for each format. The number of formats is stored in Fmt_Counts (page 116) in the Qdb_Qdbwh header.

Offset					
Dec	Hex	Bit	Type	Field	Description
0	0		CHAR(10)	Rec_Name	Record format name. The name of this particular record format for the file.
10	A		CHAR(2)	Reserved	Reserved.
12	C		BINARY(2)	Num_Of_Keys	Number of record format key fields.
14	E		CHAR(14)	Reserved	Reserved.
28	1C		BINARY(4)	Key_Info_Offset	Offset to the key field description array for this record format.

Key Field Description Array (Qdb_Qdbwhkey)

You can locate the *Qdb_Qdbwhkey* section with the offset Key_Info_Offset (page 116) in the Qdb_Qdbwhrec (page 116) array member. This is a linked list. There is a key field information array member for each key in the record format. The number of key fields is stored in Num_Of_Keys (page 116) in the Qdb_Qdbwhrec array member.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Field**	**Description**
0	0		CHAR(10)	Int_Field_Name	Internal key field name. If this is a logical format, this name is the name of the field in the logical format. If this is a physical format, this name is the same as the external field name.
10	A		CHAR(10)	Ext_Field_Name	External key field name. If this is a physical format, this is the name of the field in the physical format. If this is a logical format, this name is the name of the field in a physical format on which this format is based.
20	14		BINARY(2)	Data_Type	The data type of this key field.
22	16		BINARY(2)	Field_Len	The length of this key field.
24	18		BINARY(2)	Num_Of_Digs	The number of digits in this key field. For numeric fields, this is the number of digits. For graphic data fields, this is the number of DBCS characters the field can contain. This field is applicable only to numeric and graphic fields.
26	1A		BINARY(2)	Dec_Pos	The number of decimal positions for this key field.
28	1C	0	CHAR(1)	Qdb_ Qdbwhkattr_t	Key field attributes flags.
28	1C	0	BIT(1)	Descending	Descending/ascending sequence indicator.
28	1C	1	BIT(2)	Numeric	Numeric key field sequencing indicator.
28	1C	3	BIT(1)	Reserved	Reserved.
28	1C	4	BIT(1)	Alt_Collating	Alternative collating sequence indicator.
28	1C	5	BIT(1)	Force_Zone	Force zone sequence indicator.
28	1C	6	BIT(1)	Force_Digit	Force digit sequence indicator.
28	1C	7	BIT(1)	Statement_ Format	Key statement external or internal name indicator.
29	1D		BINARY(2)	Alt_Name_Len	Length of the alternative name. If the length of the alternative name is greater than 30, this field will be 0. The longer alternative name will have to be accessed by the Alias Name Structure (Qdb_Qddfalis).
31	1F		CHAR(30)	Alt_Name	Alternative name (Alias). If the length of the alternative name is greater than 30, this field will be blank. The longer alternative name will have to be accessed by the Alias Name Structure (Qdb_Qddfalis).
61	3D		CHAR(1)	Reserved	Reserved.
62	3E	0	CHAR(1)	Qdb_ Qdbwhkatt1_t	Additional key field attribute flags.
62	3E	0	BIT(1)	Null_Value	Allow null value (ALNULL) indicator.
62	3E	1	BIT(1)	Alt_Name_ Exists	The alternative name indicator. If the key field has an alternative name, this field will be 1, even if the length of the alternative name is greater than 30.

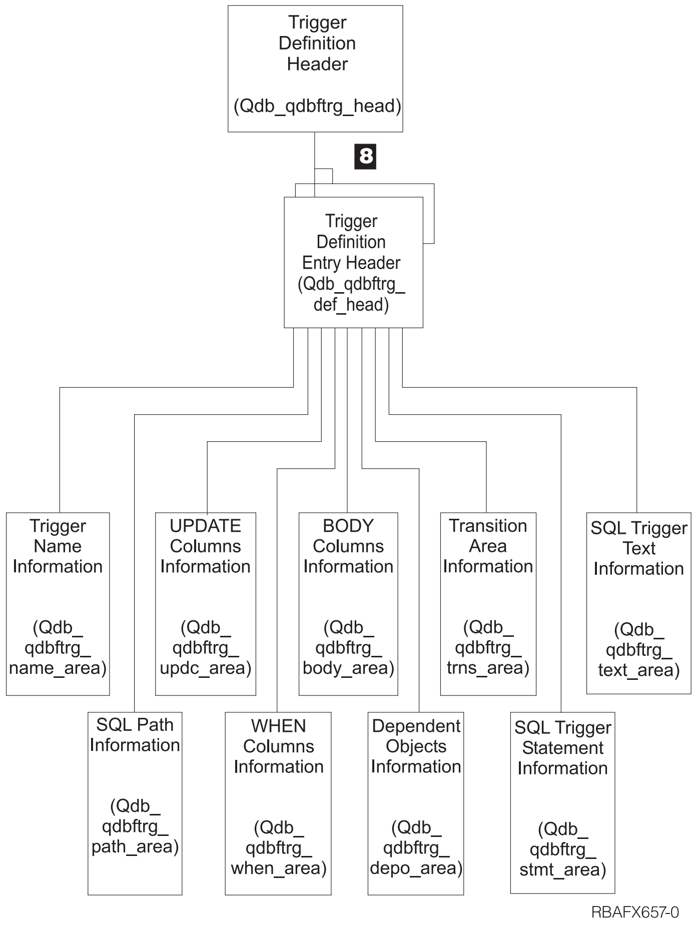
Offset		Bit	Type	Field	Description
Dec	Hex				
62	3E	2	BIT(6)	Reserved	Reserved.
63	3F		CHAR(1)	Reserved	Reserved.

FILD0400 Format (Qdb_qdbftrg_head structure)

FILD0400 provides detailed information about triggers defined for a file. FILD0400 Format (page 118) shows how this information is organized. When more than one entry can appear, the figure indicates this as in **(8)**.

Descriptions of the fields in this structure follow FILD0400 Format (page 118). The include source is supplied on the system, in the appropriate language source file, member name QDBRTVFD, in the QSYSINC library. The field names in the following tables apply only to the ILE C include. Refer to Include files and the QSYSINC Library for the names of the OPM and ILE RPG and COBOL includes.

FILD0400 Format


Trigger Definition Header (Qdb_qdbftrg_head)
8
Trigger Definition Entry Header (Qdb_qdbftrg_ def_head)
Trigger Name Information (Qdb_ qdbftrg_ name_area)
UPDATE Columns Information (Qdb_ qdbftrg_ updc_area)
BODY Columns Information (Qdb_ qdbftrg_ body_area)
Transition Area Information (Qdb_ qdbftrg_ trns_area)
SQL Trigger Text Information (Qdb_ qdbftrg_ text_area)
SQL Path Information (Qdb_ qdbftrg_ path_area)
WHEN Columns Information (Qdb_ qdbftrg_ when_area)
Dependent Objects Information (Qdb_ qdbftrg_ depo_area)
SQL Trigger Statement Information (Qdb_ qdbftrg_ stmt_area)
RBAFX657-0

Trigger Information Header (Qdb_qdbftrg_head)

This is the first structure and is located at offset zero of the returned data.

Offset		Type	Field	Description
Dec	Hex			
0	0	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Bytes_Returned	Length of the data returned in bytes.
4	4	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Bytes_Avail	Number of bytes available for the trigger information data.
8	8	CHAR(52)	Qdb_Qdbftrg_Reserved1	Reserved.
60	3C	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Num_Trgs	Number of trigger definitions.
64	40	CHAR(8)	Qdb_Qdbftrg_Reserved2	Reserved.
72	48	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Off_Ent_Num1	Offset to first trigger definition entry.
76	4C	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Off_Ins_Grp	Offset to the beginning of the insert group.
80	50	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Off_Upd_Grp	Offset to the beginning of the update group.
84	54	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Off_Del_Grp	Offset to the beginning of the delete group.
88	58	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Off_Read_Grp	Offset to the beginning of the read group.
92	5C	CHAR(28)	Qdb_Qdbftrg_Reserved36	Reserved.
120	78	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Num_Sql_Trgs	Number of SQL triggers.
124	7C	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Num_Ntv_Trgs	Number of native triggers.
128	80	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Num_Insb_Trg	Number of INSERT/BEFORE triggers.
132	84	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Num_Insa_Trg	Number of INSERT/AFTER triggers.
136	88	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Num_Delb_Trg	Number of DELETE/BEFORE triggers.
140	8C	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Num_Dela_Trg	Number of DELETE/AFTER triggers.
144	90	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Num_Updb_Trg	Number of UPDATE/BEFORE triggers.
148	94	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Num_Upda_Trg	Number of UPDATE/AFTER triggers.
152	98	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Num_Reada_Trg	Number of READ/AFTER triggers.
156	9C	CHAR(548)	Qdb_Qdbftrg_Reserved3	Reserved.

Trigger Definition Entry Header (Qdb_Qdbftrg_Def_Head)

The number of entries is defined by variable Qdb_Qdbftrg_Num_Trgs (page 119) in the trigger header, Qdb_Qdbftrg_Head (page 119). You can locate the *Qdb_Qdbftrg_Def_Head* section with the offset Qdb_Qdbftrg_Def_Off_Ent_Num1 (page 119) in the trigger header, Qdb_Qdbftrg_Head.

Offset				
Dec	Hex	Type	Field	Description
0	0	CHAR(20)	Qdb_Qdbftrg_Reserved4	Reserved.
20	14	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Def_Len	Length of the entire trigger definition. This includes all structures. This length added to the pointer to this entry gets you to the next trigger definition entry.
24	18	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Head_Len	Length of the trigger definition header Qdb_Qdbftrg_Def_Head (page 119).
28	1C	CHAR(52)	Qdb_Qdbftrg_Reserved5	Reserved.
80	50	CHAR(10)	Qdb_Qdbftrg_Def_Pgm	Trigger program name.
90	5A	CHAR(10)	Qdb_Qdbftrg_Def_Lib	Trigger program library.
100	64	CHAR(4)	Qdb_Qdbftrg_Reserved6	Reserved.
104	68	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Def_Ord_ Num	Trigger ordinal number.
108	6C	CHAR(1)	Qdb_Qdbftrg_Def_State	Trigger state. *'E'* Trigger is enabled. *'D'* Trigger is disabled.
109	6D	CHAR(1)	Qdb_Qdbftrg_Def_Operative	Trigger is operative. *'O'* Trigger is operative. *'I'* Trigger is inoperative.
110	6E	CHAR(1)	Qdb_Qdbftrg_Def_Type	Trigger type. *'N'* Native/System (added using ADDPFTRG). *'S'* SQL (added using CREATE TRIGGER).
111	6F	CHAR(1)	Qdb_Qdbftrg_Def_Mode	Trigger mode. Used only when the trigger type QDBFTRG_DEF_TYPE (page 119) is set to an SQL trigger. The mode is used to determine the I/O model used during the trigger program execution. *'00'X* Not applicable. *'01'X* DB2 SQL. *'02'X* DB2 Row.
112	70	CHAR(1)	Qdb_Qdbftrg_Def_Orient	Trigger orientation. *'R'* Row trigger *'C'* Column trigger
113	71	CHAR(1)	Qdb_Qdbftrg_Def_Time	Trigger time. *'1'* After *'2'* Before
114	72	CHAR(1)	Qdb_Qdbftrg_Def_Event	Trigger event. *'1'* Insert *'2'* Delete *'3'* Update *'4'* Read

Offset		Type	Field	Description
Dec	Hex			
115	73	CHAR(45)	Qdb_Qdbftrg_Reserved7	Reserved.
160	A0	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Def_Off_ Trg_Name	Offset to the trigger name structure Qdb_Qdbftrg_Name_Area (page 123).
164	A4	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Reserved8	Reserved.
168	A8	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Def_Off_Sql_Path	Offset to the SQL path structure Qdb_Qdbftrg_Path_Area (page 123).
172	AC	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Def_Off_Upd_Cols	Offset to the update columns structure Qdb_Qdbftrg_Updc_Area (page 124).
176	B0	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Def_Off_ When_Cols	Offset to the structure containing the list of columns referenced in the WHEN condition Qdb_Qdbftrg_When_Area (page 125).
180	B4	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Def_Off_ Body_Cols	Offset to the structure containing the list of columns referenced in the trigger body Qdb_Qdbftrg_Body_Area (page 127).
184	B8	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Def_Off_ Dep_Objs	Offset to the structure containing the list of dependent objects referenced in the trigger body Qdb_Qdbftrg_Depo_Area (page 128).
188	BC	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Def_Off_ Transition	Offset to the structure containing the transition tables, Qdb_Qdbftrg_Trns_Area (page 130).
192	C0	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Def_Off_ Trg_Stmt	Offset to the structure containing the CREATE TRIGGER statement rQdb_Qdbftrg_Stmt_Area (page 131).
196	C4	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Def_Off_ Trg_Long	Offset to the structure containing the trigger long comment Qdb_Qdbftrg_Long_Area (page 132).
200	C8	CHAR(64)	Qdb_Qdbftrg_Reserved9	Reserved.
264	108	CHAR(1)	Qdb_Qdbftrg_Def_ Updcond	Trigger update condition. For system triggers only (Qdb_Qdbftrg_Def_Type = 'N'). This field is valid for the UPDATE event only. This field is ignored for INSERT or DELETE or READ events. *'1'* Always calls the trigger when updating the file. *'2'* Only calls the trigger when the updated values are changed.
265	109	CHAR(1)	Qdb_Qdbftrg_Def_ Allow_Repchg	Allow repeated change. *X'00'* *NO (Repeated change not allowed) *X'01'* *YES (Repeated change allowed)
266	10A	CHAR(1)	Qdb_Qdbftrg_Def_ Threadsafe	Threadsafe indicator. *X'00'* *UNKNOWN (Threadsafe status is not known) *X'01'* *NO (Not threadsafe) *X'10'* *YES (Threadsafe)

Offset				
Dec	Hex	Type	Field	Description
267	10B	CHAR(1)	Qdb_Qdbftrg_Def_ Multijob	Multithreaded job action indicator *X'00'* *SYSVAL (default) *X'01'* *MSG (Run, diagnostic) *X'10'* *NORUN (Escape) *X'11'* *RUN (Run, no message)
268	10C	CHAR(1)	Qdb_Qdbftrg_Def_Old_Tvar	Old correlation variable indicator. Only applies to SQL triggers. *X'00'* No *X'01'* Yes
269	10D	CHAR(1)	Qdb_Qdbftrg_Def_New_Tvar	New correlation variable indicator. Only applies to SQL triggers. *X'00'* No *X'01'* Yes
270	10E	CHAR(1)	Qdb_Qdbftrg_Def_Old_Ttable	Old transition table indicator. *X'00'* No *X'01'* Yes
271	10F	CHAR(1)	Qdb_Qdbftrg_Def_New_Ttable	New transition table indicator *X'00'* No *X'01'* Yes
272	110	CHAR(1)	Qdb_Qdbftrg_Def_Self_Ref	Self-referencing indicator. Indicates whether or not the user explicitly specified this file's name in the trigger body. *X'00'* Not self-referencing. *X'01'* Self-referencing.
273	111	CHAR(13)	Qdb_Qdbftrg_Def_Crt_Ts	Trigger creation timestamp. The format is CYYMMDDHHMMSS.
286	11E	CHAR(10)	Qdb_Qdbftrg_Def_Crt_User	User profile that created the trigger.
296	11F	CHAR(10)	Qdb_Qdbftrg_Def_Pgm_Owner	User profile that owns the trigger program. For SQL triggers only.
306	132	BIN(4) UNSIGNED	Qdb_Qdbftrg_Def_Trg_Ccsid	CCSID of the CREATE TRIGGER statement.
310	136	CHAR(1)	Qdb_Qdbftrg_Reserved34	Reserved.
311	137	CHAR(1)	Qdb_Qdbftrg_Def_Mod_Tvar	The trigger contains a SET statement that modifies the new correlation variable. (Indicates whether or not update authority is required to the table.) For SQL *BEFORE *UPDATE triggers only. *X'00'* No. *X'01'* Yes.
312	138	CHAR(152)	Qdb_Qdbftrg_Reserved10	Reserved.

Trigger Definition Name Structure (Qdb_Qdbftrg_Name_Area)

You can locate the *Qdb_Qdbftrg_Name_Area* section with the offset Qdb_Qdbftrg_Def_Off_Trg_Name (page 121) in the Qdb_Qdbftrg_Def_Head (page 119) section.

Offset				
Dec	**Hex**	**Type**	**Field**	**Description**
0	0	CHAR(20)	Qdb_Qdbftrg_Reserved11	Reserved.
20	14	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Name_ Area_Tot_Len	Total length of the trigger name area Qdb_Qdbftrg_Name_Area (page 123).
24	18	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Name_Lib_Len	Length of the trigger library name.
28	1C	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Name_Len	Length of the trigger program name.
32	20	CHAR(1)	Qdb_Qdbftrg_Name_Sysgen	System-generated trigger name indicator. *X'00'* The user specified the trigger name. *X'01'* Originally, the user generated the name, but a name collision occurred and the system generated a new name. *X'02'* The system generated the trigger name.
33	21	CHAR(1)	Qdb_Qdbftrg_Name_Delim	Delimited name indicator. If the trigger name is delimited, it will contain double quotes. *X'00'* The trigger name is not delimited. *X'01'* The trigger name is delimited.
34	22	CHAR(1)	Qdb_Qdbftrg_Lib_Delim	Delimited library name indicator. If the trigger library name is delimited, it will contain double quotes. *X'00'* The trigger library name is not delimited. *X'01'* The trigger library name is delimited.
35	23	CHAR(1)	Qdb_Qdbftrg_Name_Type	Trigger naming convention. *X'00'* System naming. *X'01'* SQL naming.
36	24	CHAR(58)	Qdb_Qdbftrg_Reserved12	Reserved.
94	5E	CHAR(268)	Qdb_Qdbftrg_Name_Qual	Qualified trigger name. The trigger name and library name are in two parts. Part 1 is the trigger library name, which is padded to 10 characters, if necessary, with blanks. Qdb_Qdbftrg_Name_Lib_Len (page 123) defines the length of the trigger library name. Part 2 is the trigger name. Qdb_Qdbftrg_Name_Len (page 123) defines the length of the trigger name.

SQL Path Structure (Qdb_Qdbftrg_Path_Area)

You can locate the *Qdb_Qdbftrg_Path_Area* section with the offset Qdb_Qdbftrg_Def_Off_Sql_Path (page 121) in the Qdb_Qdbftrg_Def_Head (page 119) section. This structure is for SQL triggers only.

Offset		Type	Field	Description
Dec	Hex			
0	0	CHAR(20)	Qdb_Qdbftrg_Reserved13	Reserved.
20	14	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Path_Tot_Len	Total length of the SQL path area Qdb_Qdbftrg_Path_Area (page 123).
24	18	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Path_Len	Length of the SQL path structure Qdb_Qdbftrg_Path (page 124).
28	1C	CHAR(36)	Qdb_Qdbftrg_Reserved14	Reserved.
64	40	CHAR(*)	Qdb_Qdbftrg_Path	SQL path. See DB2 UDB for iSeries SQL Reference CURRENT PATH special register for information on the format of this structure.

UPDATE Columns Structure (Qdb_Qdbftrg_Updc_Area)

You can locate the *Qdb_Qdbftrg_Updc_Area* section with the offset Qdb_Qdbftrg_Def_Off_Upd_Col (page 121) in the Qdb_Qdbftrg_Def_Head (page 119) section. This structure is for SQL triggers only.

Offset		Type	Field	Description
Dec	Hex			
0	0	CHAR(20)	Qdb_Qdbftrg_Reserved15	Reserved.
20	14	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Updc_Tot_Len	Total length of the update columns area Qdb_Qdbftrg_Updc_Area (page 124).
24	18	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Updc_Num_Cols	Number of columns in the list Qdb_Qdbftrg_Updc_List_Struc (page 124).
28	1C	CHAR(52)	Qdb_Qdbftrg_Reserved16	Reserved.
80	50	CHAR(*)	Qdb_Qdbftrg_Updc_List_Struc	Update column list structure.

UPDATE Columns Entry Structure (Qdb_Qdbftrg_Updc_List_Ent)

The *Qdb_Qdbftrg_Updc_List_Ent* section maps an entry in the structure Qdb_Qdbftrg_Updc_List_Struc (page 124).

Offset		Type	Field	Description
Dec	Hex			
0	0	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Updc_Le_Len	Update column list entry length. Addressability to this entry plus Qdb_Qdbftrg_Updc_Le_Len (page 124) gets addressability to the next entry in this structure.
4	4	CHAR(10)	Qdb_Qdbftrg_Updc_Le_ Short_Name	Short name of the column.
14	E	CHAR(1)	Qdb_Qdbftrg_Updc_Le_ Short_Del	Short name is delimited indicator. *X'00'* Name is not delimited. *X'01'* Name is delimited.
15	F	CHAR(1)	Qdb_Qdbftrg_Updc_Le_ Long_Del	Long name is delimited indicator. *X'00'* Name is not delimited. *X'01'* Name is delimited.

Offset		Type	Field	Description
Dec	Hex			
16	10	CHAR(1)	Qdb_Qdbftrg_Updc_Le_ Long_Same	Short name and long name are the same indicator. *X'00'* Names are different. *X'01'* Names are the same.
17	11	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Updc_Le_ Long_Len	Length of the long name Qdb_Qdbftrg_Updc_Le_Long_Name (page 125).
21	15	CHAR(27)	Qdb_Qdbftrg_Reserved17	Reserved.
48	30	CHAR(*)	Qdb_Qdbftrg_Updc_Le_ Long_Name	Long name of the column.

WHEN Columns Structure (Qdb_Qdbftrg_When_Area)

You can locate the *Qdb_Qdbftrg_When_Area* section with the offset Qdb_Qdbftrg_Def_Off_When_Cols (page 121) in the Qdb_Qdbftrg_Def_Head (page 119) section. This structure is for SQL triggers only.

Offset		Type	Field	Description
Dec	Hex			
0	0	CHAR(20)	Qdb_Qdbftrg_Reserved18	Reserved.
20	14	BINARY(4) UNSIGNED	Qdb_Qdbftrg_When_Tot_Len	Total length of the WHEN columns area Qdb_Qdbftrg_When_Area (page 125).
24	18	BINARY(4) UNSIGNED	Qdb_Qdbftrg_When_Num_Cols	Number of columns in the list Qdb_Qdbftrg_When_List_Struc (page 125).
28	1C	CHAR(1)	Qdb_Qdbftrg_When_Self_Ref	Whether columns in the WHEN list belong to this file. *X'00'* All columns belong to the ON table. *X'01'* Some of the columns belong to the ON table. *X'02'* None of the columns belong to the ON table.
29	1D	CHAR(51)	Qdb_Qdbftrg_Reserved19	Reserved.
80	50		Qdb_Qdbftrg_When_Array	WHEN column array.

WHEN Columns Entry Structure (Qdb_Qdbftrg_When_Array)

This Qdb_Qdbftrg_When_Array (page 125) section maps an entry in the structure Qdb_Qdbftrg_When_Array. This structure is for SQL triggers only.

Offset		Type	Field	Description
Dec	Hex			
0	0	BINARY(4) UNSIGNED	Qdb_Qdbftrg_When_Col_Off	WHEN column name offset into the WHEN portion of the CREATE TRIGGER statement Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) .
4	4	BINARY(4) UNSIGNED	Qdb_Qdbftrg_When_Col_Len	Length of the column name in the CREATE TRIGGER statement Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) .

Offset				
Dec	Hex	Type	Field	Description
8	8	BINARY(4) UNSIGNED	Qdb_Qdbftrg_When_File_Off	Offset to the column's file name in the CREATE TRIGGER statement Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) relative to the start of the CREATE TRIGGER statement.
12	C	BINARY(4) UNSIGNED	Qdb_Qdbftrg_When_File_Len	Length of the column's file name in the CREATE TRIGGER statement Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) .
16	10	BINARY(4) UNSIGNED	Qdb_Qdbftrg_When_Lib_Off	Offset to the column's library name in the CREATE TRIGGER statement Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) relative to the start of the CREATE TRIGGER statement.
20	14	BINARY(4) UNSIGNED	Qdb_Qdbftrg_When_Lib_Len	Length of the column's library name in the trigger string Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) .
24	18	CHAR(1)	Qdb_Qdbftrg_When_1st_Entry	Whether the column name is the first in the list of entries. *X'00'* Column is not the first in the list. *X'01'* Column name is the first in the list.
25	19	CHAR(1)	Qdb_Qdbftrg_When_This_File	Whether the column name is in this file. *X'00'* Column is not in this file. *X'01'* Column name is in this file.
26	1A	CHAR(1)	Qdb_Qdbftrg_When_Col_Long	Whether the column name is a short or long name. *X'00'* Column name is short name. *X'01'* Column name is long name.
27	1B	CHAR(1)	Qdb_Qdbftrg_When_Col_Del	Whether the column name is a delimited name. *X'00'* Column name is is not delimited. *X'01'* Column name is delimited.
28	1C	CHAR(1)	Qdb_Qdbftrg_When_File_Long	Whether the column's file name is a short or long name. *X'00'* File name is short name. *X'01'* File name is long name.
29	1D	CHAR(1)	Qdb_Qdbftrg_When_File_Del	Whether the column's file name is a delimited name. *X'00'* File name is not delimited. *X'01'* File name is delimited.
30	1E	CHAR(1)	Qdb_Qdbftrg_When_Lib_Long	Whether the column's library name is a short or long name. *X'00'* Library name is short name. *X'01'* Library name is long name.
31	1F	CHAR(1)	Qdb_Qdbftrg_When_Lib_Del	Whether the column's library name is a delimited name. *X'00'* Columns name is not delimited. *X'01'* Column name is delimited.

Offset				
Dec	Hex	Type	Field	Description
32	20	CHAR(32)	Qdb_Qdbftrg_Reserved20	Reserved.

BODY Columns Structure (Qdb_Qdbftrg_Body_Area)

You can locate this *Qdb_Qdbftrg_Body_Area* section with the offset Qdb_Qdbftrg_Def_Off_Body_Cols (page 121) in the Qdb_Qdbftrg_Def_Head (page 119) section. This structure is for SQL triggers only.

Offset				
Dec	Hex	Type	Field	Description
0	0	CHAR(20)	Qdb_Qdbftrg_Reserved21	Reserved.
20	14	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Body_Tot_Len	Total length of the BODY columns area Qdb_Qdbftrg_Body_Area (page 127).
24	18	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Body_Num_Cols	Number of columns in the list Qdb_Qdbftrg_Body_List_Struc (page 127).
28	1C	CHAR(1)	Qdb_Qdbftrg_Body_Self_Ref	Whether columns in the body list belong to this file. *X'00'* All self-referencing. *X'01'* Some self-referencing. *X'10'* Not self-referencing.
29	1D	CHAR(51)	Qdb_Qdbftrg_Reserved22	Reserved.
80	50		Qdb_Qdbftrg_Body_Array	Array used to access the list of BODY referenced columns.

BODY Columns Entry Structure (Qdb_Qdbftrg_Body_Array)

The *Qdb_Qdbftrg_Body_Array* section maps an entry in the structure Qdb_Qdbftrg_Body_Array (page 127). This structure is for SQL triggers only.

Offset				
Dec	Hex	Type	Field	Description
0	0	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Body_Col_Off	BODY column name offset into the BODY portion of the CREATE TRIGGER statement Qdb_Qdbftrg_Stmt_Crt_Trg (page 132).
4	4	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Body_Col_Len	Length of the column name in the CREATE TRIGGER statement Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) .
8	8	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Body_File_Off	Offset to the column's file name in the CREATE TRIGGER statement Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) relative to the start of the CREATE TRIGGER statement.
12	C	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Body_File_Len	Length of the column's file name in the CREATE TRIGGER statement Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) .
16	10	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Body_Lib_Off	Offset to the column's library name in the CREATE TRIGGER statement Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) relative to the start of the CREATE TRIGGER statement.

Offset		Type	Field	Description
Dec	Hex			
20	14	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Body_Lib_Len	Length of the column's library name in the CREATE TRIGGER statement Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) .
24	18	CHAR(1)	Qdb_Qdbftrg_Body_1st_Entry	Whether the column name is the first in the list of entries. *X'00'* Column is not first in the list. *X'01'* Column name is first in the list.
25	19	CHAR(1)	Qdb_Qdbftrg_Body_This_File	Whether the column name is in this file. *X'00'* Column is not in this file. *X'01'* Column name is in this file.
26	1A	CHAR(1)	Qdb_Qdbftrg_Body_Col_Long	Whether the column name is a short or long name. *X'00'* Column name is short name. *X'01'* Column name is long name.
27	1B	CHAR(1)	Qdb_Qdbftrg_Body_Col_Del	Whether the column name is a delimited name. *X'00'* Column name is is not delimited. *X'01'* Column name is delimited.
28	1C	CHAR(1)	Qdb_Qdbftrg_Body_File_Long	Whether the column's file name is a short or long name. *X'00'* File name is short name. *X'01'* File name is long name.
29	1D	CHAR(1)	Qdb_Qdbftrg_Body_File_Del	Whether the column's file name is a delimited name. *X'00'* File name is not delimited. *X'01'* File name is delimited.
30	1E	CHAR(1)	Qdb_Qdbftrg_Body_Lib_Long	Whether the column's library name is a short or long name. *X'00'* Library name is short name. *X'01'* Library name is long name.
31	1F	CHAR(1)	Qdb_Qdbftrg_Body_Lib_Del	Whether the column's library name is a delimited name. *X'00'* Columns name is not delimited. *X'01'* Column name is delimited.
32	20	CHAR(32)	Qdb_Qdbftrg_Reserved23	Reserved.

Dependent Objects Structure (Qdb_Qdbftrg_Depo_Area)

You can locate the *Qdb_Qdbftrg_Depo_Area* section with the offset Qdb_Qdbftrg_Def_Off_Dep_Objs (page 121) in the Qdb_Qdbftrg_Def_Head (page 119) section. This structure is for SQL triggers only.

Offset				
Dec	**Hex**	**Type**	**Field**	**Description**
0	0	CHAR(20)	Qdb_Qdbftrg_Reserved24	Reserved.
20	14	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Depo_Tot_Len	Total length of the dependent objects area Qdb_Qdbftrg_Depo_Area (page 128).
24	18	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Depo_Num_Off	Number of dependent object offsets in Qdb_Qdbftrg_Depo_Array (page 129). These are offsets into the CREATE TRIGGER statement Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) that will position you to a dependent object of the type Qdb_Qdbftrg_Depo_Type (page 129).
28	1C	CHAR(1)	Qdb_Qdbftrg_Depo_Self_Ref	Whether this file is referenced at least once somewhere in the WHEN or BODY. This is a self-referencing dependency. *X'00'* Not self-referencing. *X'01'* Is self-referencing.
29	1D	CHAR(51)	Qdb_Qdbftrg_Reserved25	Reserved.
80	50		Qdb_Qdbftrg_Depo_Array	Update column list structure.

Dependent Objects Entry Structure (Qdb_Qdbftrg_Depo_Array)

The *Qdb_Qdbftrg_Depo_Array* section maps an entry in the structure Qdb_Qdbftrg_Depo_Array (page 129). This structure is for SQL triggers only.

Offset				
Dec	**Hex**	**Type**	**Field**	**Description**
0	0	CHAR(2)	Qdb_Qdbftrg_Depo_ Type	Type of dependent object. *'TB'* Table *'PF'* Physical File *'VW'* View *'LF'* Logical File *'IX'* Index *'UF'* User Defined Function *'UT'* User Defined Type *'PR'* Procedure *'AL'* Alias
2	2	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Depo_ Off	Offset to the dependent object relative to the beginning of the CREATE TRIGGER string Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) .
6	6	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Depo_ Len	Length of the dependent object in the CREATE TRIGGER string Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) .
10	A	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Depo_ Lib_Off	Offset to the qualifying library name of the dependent object. Offset relative from the start of Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) . A length of 0 indicates no qualifying library.

Offset				
Dec	Hex	Type	Field	Description
14	E	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Depo_ Lib_Len	Length of the qualifying library name of the dependent object. A length of 0 indicates there is no qualifying library.
18	12	CHAR(1)	Qdb_Qdbftrg_Depo_ 1st_Entry	Whether the object name is the first in the list of entries. Criteria is based on object name, library name, and object type. *X'00'* Object is 2nd or greater occurance in the list. *X'01'* Object is the first in the list.
19	13	CHAR(1)	Qdb_Qdbftrg_Depo_ This_File	Whether the object is this file. *X'00'* Object is not this file. *X'01'* Object name is this file.
20	14	CHAR(1)	Qdb_Qdbftrg_Depo_ Obj_Long	Whether the object name is is a short or long name. *X'00'* Object name is short name. *X'01'* Object name is long name.
21	15	CHAR(1)	Qdb_Qdbftrg_Depo_ Obj_Del	Whether the object name is a delimited name. *X'00'* Object name is is not delimited. *X'01'* Object name is delimited.
22	16	CHAR(1)	Qdb_Qdbftrg_Depo_ Lib_Long	Whether the library's name is a short or long name. *X'00'* Library name is short name. *X'01'* Library name is long name.
23	17	CHAR(1)	Qdb_Qdbftrg_Depo_ Lib_Del	Whether the library's name is a delimited name. *X'00'* Library name is not delimited. *X'01'* Library name is delimited.
24	18	CHAR(40)	Qdb_Qdbftrg_ Reserved26	Reserved.

Transition Area Structure (Qdb_Qdbftrg_Trns_Area)

You can locate the *Qdb_Qdbftrg_Trns_Area* section with the offset Qdb_Qdbftrg_Def_Off_Transition (page 121) in the Qdb_Qdbftrg_Def_Head (page 119) section. This structure is for SQL triggers only.

Offset				
Dec	Hex	Type	Field	Description
0	0	CHAR(20)	Qdb_Qdbftrg_Reserved27	Reserved.
20	14	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Trns_Tot_Len	Total length of the transition area Qdb_Qdbftrg_Trns_Area (page 130).
24	18	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Trns_Oldvar_Len	Old correlation variable name length.
28	1C	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Trns_Newvar_Len	New correlation variable name length.
32	20	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Trns_Oldtbl_Len	Old transition table name length.

Offset				
Dec	Hex	Type	Field	Description
36	24	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Trns_Newtbl_Len	New transition table name length.
40	28	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Trns_Oldvar_Off	Old correlation variable name offset.
44	2C	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Trns_Newvar_Off	New correlation variable name offset.
48	30	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Trns_Oldtbl_Off	Old transition table name offset.
52	34	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Trns_Newtbl_Off	New transition table name offset.
56	38	CHAR(1)	Qdb_Qdbftrg_Trns_Oldvar_Del	Whether the old correlation variable name is delimited. *X'00'* Name is not delimited. *X'01'* Name is delimited.
57	39	CHAR(1)	Qdb_Qdbftrg_Trns_Newvar_Del	Whether the new correlation variable name is delimited. *X'00'* Name is not delimited. *X'01'* Name is delimited.
58	3A	CHAR(1)	Qdb_Qdbftrg_Trns_Oldtbl_Del	Whether the old table name is delimited. *X'00'* Name is not delimited. *X'01'* Name is delimited.
59	3B	CHAR(1)	Qdb_Qdbftrg_Trns_Newtbl_Del	Whether the new table name is delmited. *X'00'* Name is not delimited. *X'01'* Name is delimited.
60	3C	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Trns_Names_Len	Length of the transition names Qdb_Qdbftrg_Trns_Names (page 131).
64	40	CHAR(48)	Qdb_Qdbftrg_Reserved28	Reserved.
112	70	CHAR(*)	Qdb_Qdbftrg_Trns_Names	Old/new transition variable/table names.

Trigger Statement Area (Qdb_Qdbftrg_Stmt_Area)

You can locate the *Qdb_Qdbftrg_Stmt_Area* section with the offset Qdb_Qdbftrg_Def_Off_Trg_Stmt (page 121) in the Qdb_Qdbftrg_Def_Head (page 119) section. This structure is for SQL triggers only.

Offset				
Dec	Hex	Type	Field	Description
0	0	CHAR(20)	Qdb_Qdbftrg_Reserved29	Reserved.
20	14	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Stmt_Tot_Len	Total length of the statement area.
24	18	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Stmt_Onfile_Off	Offset to the user-specified file table name in the CREATE TRIGGER statement Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) .
28	1C	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Stmt_Onfile_Len	Length of the qualified library file name of the ON file/TABLE name, including the period, in Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) .

Offset		Type	Field	Description
Dec	Hex			
32	20	CHAR(1)	Qdb_Qdbftrg_Stmt_Onfile_Del	Whether the ON table/file name is delimited. *X'00'* Name is not delimited. *X'01'* Name is delimited.
33	21	CHAR(1)	Qdb_Qdbftrg_Stmt_Onlib_Del	Whether the ON library name is delimited. *X'00'* Name is not delimited. *X'01'* Name is delimited.
34	22	CHAR(1)	Qdb_Qdbftrg_Stmt_Onfile_Long	Whether the ON table/file name is a long name. *X'00'* Name is not a long name. *X'01'* Name is a long name.
35	23	CHAR(1)	Qdb_Qdbftrg_Reserved30	Reserved.
36	24	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Stmt_Crt_Trg_Len	Length of the SQL CREATE TRIGGER string in variable Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) .
40	28	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Stmt_When_Len	Length of the WHEN clause for the SQL CREATE TRIGGER string in variable Qdb_Qdbftrg_Crt_Trg (page 132).
44	2C	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Stmt_When_Off	Offset to the WHEN clause of the SQL CREATE TRIGGER string in variable Qdb_Qdbftrg_Stmt_Crt_Trg (page 132) .
48	30	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Stmt_Body_Len	Length of the BODY portion of the SQL CREATE TRIGGER string in variable Qdb_Qdbftrg_Stmt_Crt_Trg.
52	34	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Stmt_Body_Off	Offset to the BODY portion of the SQL CREATE TRIGGER string in variable Qdb_Qdbftrg_Stmt_Crt_Trg.
56	38	CHAR(56)	Qdb_Qdbftrg_Reserved31	Reserved.
112	70	CHAR(*)	Qdb_Qdbftrg_Stmt_Crt_Trg	SQL CREATE TRIGGER string.

Trigger Long Comment Area (Qdb_Qdbftrg_Long_Area)

You can locate the *Qdb_Qdbftrg_Long_Area* section with the offset Qdb_Qdbftrg_Def_Off_Trg_Long (page 121) in the Qdb_Qdbftrg_Def_Head (page 119) section.

Offset		Type	Field	Description
Dec	Hex			
0	0	CHAR(20)	Qdb_Qdbftrg_Reserved32	Reserved.
20	14	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Long_Tot_Len	Total length of the long comment area Qdb_Qdbftrg_Long_Area (page 132).
24	18	BINARY(4) UNSIGNED	Qdb_Qdbftrg_Long_Len	Length of the trigger long comment that is located in variable Qdb_Qdbftrg_Long_Comment (page 132).
28	1C	BINARY(2) UNSIGNED	Qdb_Qdbftrg_Long_Ccsid	CCSID of the long comment in Qdb_Qdbftrg_Long_Comment (page 132).
30	1E	CHAR(34)	Qdb_Qdbftrg_Reserved33	Reserved.
64	40	CHAR(*)	Qdb_Qdbftrg_Long_Comment	Trigger long comment.

Usage Notes

In multithreaded jobs, this API is not threadsafe and fails for distributed data management (DDM) files of type *SNA.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3CF1 E	Error code parameter is not valid.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C21 E	Format name &1 is not valid.
CPF3C22 E	Cannot get information about file &1.
CPF3C23 E	Object &1 is not a database file.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C25 E	Value &1 for file override parameter is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3021 E	File &1 not allowed with SYSTEM(*RMT).
CPF3025 E	File &1 not allowed with SYSTEM(*LCL).
CPF325F E	Conversion of the text failed.
CPF327A E	Value &1 for format type parameter is not valid.
CPF3270 E	Keyed file operation not allowed for file &1.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R2

Top | "Database and File APIs," on page 1 | APIs by category

Retrieve Display File Description (QDFRTVFD) API

Required Parameter Group:

1	Receiver variable
Output	Char(*)
2	Length of receiver variable
Input	Binary(4)
3	Format name
Input	Char(8)
4	Qualified file name
Input	Char(20)
5	Error code
Output	Char(*)

Default Public Authority: *USE

Threadsafe: No

The Retrieve Display File Description (QDFRTVFD) API allows you to get specific information about the data description specifications (DDS) definition used to create a display file.

If the returned data does not fill the receiver variable, the contents of the remainder of the variable are not changed.

Authorities and Locks

Library Authority
*USE

File Authority
*OBJOPR

File Lock
*SHRNUP

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)

The receiver variable that receives the information requested. You can specify the size of the area smaller than the format requested as long as you specify the length of receiver variable parameter correctly. As a result, the API returns only the data the area can hold.

Length of receiver variable
INPUT; BINARY(4)

The length of the receiver variable. If the data available is larger than the length of the receiver variable, the result is truncated. The minimum length is 8 bytes. The actual length of the structure is returned in variable WDFFSIZE in structure QDFFBASE (see the “Base File Section (QDFFBASE)” on page 137).

Format name
INPUT; CHAR(8)

The content of the information to be returned about the specified display file. You can use the following format name:

DSPF0100	Display file information

See “Format DSPF0100” on page 135 for a description of these formats.

Qualified file name
INPUT; CHAR(20)

The name of the file about which the information is to be extracted and the library in which it is located. The first 10 characters contain the file name. The second 10 characters contain the library name.

The special values for the library name follow:

**CURLIB*	The job’s current library
**LIBL*	The library list

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Format DSPF0100

Format DSPF0100 provides detailed information about how display files are built. The various structures that comprise the display file information format are organized in the following manner:

- Base file formats (see “Base File Formats” on page 137)
- File formats (see “File Formats” on page 138)
- Record formats (see “Record Formats” on page 142)
- Field formats (see “Field Formats” on page 153)
- Keyword formats (see “Keyword Formats” on page 159)
- Where-used formats (see “Where-Used Formats” on page 206)

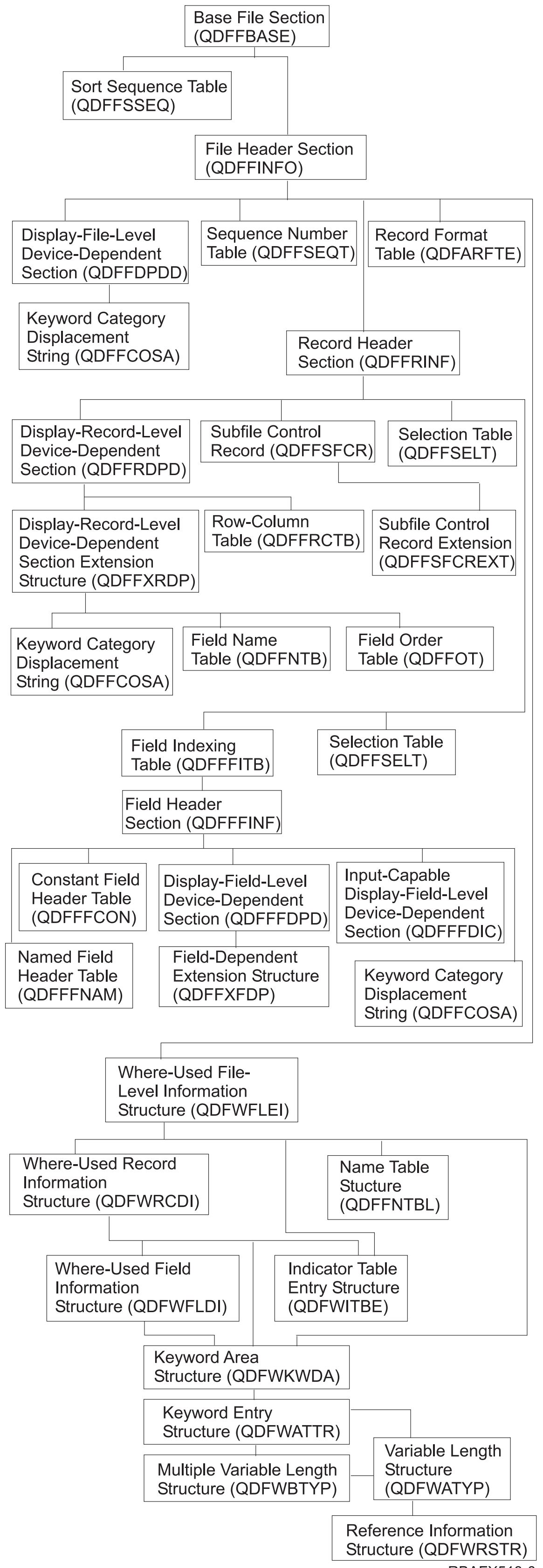
The structures for each format follow DSPF0100 Format (page 135). The structures include the variable names, field information, and offsets. Unlike many APIs, which use an offset from the beginning of the variable, most QDFRTVFD offsets are relative to the start of a base structure. To determine how to arrive at the data, see the introduction to each structure.

The use of the term **optioned** in the tables refers to an indicator that controls whether the DDS keyword is in effect or not. For more information about option indicators, see Conditioning for display files (positions 7 through 16).

The asterisk (*) in the *Variable Name* column represents a reserved field. No variable is associated with these reserved fields.

DSPF0100 Format (page 135) provides an overview of format DSPF0100 by showing how this information is organized. The abbreviated names in the figure correspond to the structure names of the tables. The formats are shown by section (for example, base file, file header, record header, and so forth). The keyword formats do not appear in the figure.

DSPF0100 Format



Base File Section (QDFFBASE)
Sort Sequence Table (QDFFSSEQ)
File Header Section (QDFFINFO)
Display-File-Level Device-Dependent Section (QDFFDPDD)
Sequence Number Table (QDFFSEQT)
Record Format Table (QDFARFTE)
Keyword Category Displacement String (QDFFCOSA)
Record Header Section (QDFFRINF)
Display-Record-Level Device-Dependent Section (QDFFRDPD)
Subfile Control Record (QDFFSFCR)
Selection Table (QDFFSELT)
Display-Record-Level Device-Dependent Section Extension Structure (QDFFXRDP)
Row-Column Table (QDFFRCTB)
Subfile Control Record Extension (QDFFSFCREXT)
Keyword Category Displacement String (QDFFCOSA)
Field Name Table (QDFFNTB)
Field Order Table (QDFFOT)
Field Indexing Table (QDFFFITB)
Selection Table (QDFFSELT)
Field Header Section (QDFFFINF)
Constant Field Header Table (QDFFFCON)
Display-Field-Level Device-Dependent Section (QDFFFDPD)
Input-Capable Display-Field-Level Device-Dependent Section (QDFFFDIC)
Named Field Header Table (QDFFFNAM)
Field-Dependent Extension Structure (QDFFXFDP)
Keyword Category Displacement String (QDFFCOSA)
Where-Used File-Level Information Structure (QDFWFLEI)
Where-Used Record Information Structure (QDFWRCDI)
Name Table Stucture (QDFFNTBL)
Where-Used Field Information Structure (QDFWFLDI)
Indicator Table Entry Structure (QDFWITBE)
Keyword Area Structure (QDFWKWDA)
Keyword Entry Structure (QDFWATTR)
Variable Length Structure (QDFWATYP)
Multiple Variable Length Structure (QDFWBTYP)
Reference Information Structure (QDFWRSTR)
RBAFX513-0

Base File Formats

The base file formats follow.

Base File Section (QDFFBASE)

Base file structure. This is the first structure and is located at offset zero of the returned data.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(31)	WDFFRETN	Length of the returned data.
4	4		BIN(31)	WDFFSIZE	Size of the display file description.
8	8		BIN(15)	WDFFINOF	Displacement to file header section (see structure QDFFINFO, "File Header Section (QDFFINFO)" on page 138).
10	A		BIN(15)	WDFFRCS	Number of record formats specified. This number includes internally generated record formats.
12	C		CHAR(1)	WDFFDPAT	Display attribute bits.
12	C	0	BIT(1)	WDFFSEPI	If on, INDARA keyword is specified.
12	C	1	BIT(1)	WDFFDESF	If on, ERRSFL keyword is specified. **Note:** The ERRSFL keyword generates additional internal records (*ERRSFL).
12	C	2	BIT(6)	*	Reserved.
13	D		BIN(15)	WDFFSCR	Number of valid file screen sizes (see structure QDFFSCRA, "Screen Size Table (QDFFSCRA)").
15	F		BIN(15)	WDFFSRSQ	Displacement to sort sequence table (see structure QDFFSSEQ, "Sort Sequence Table (QDFFSSEQ)").
17	11		CHAR(2)	WDFFACCSID	CCSID of source member used to create the device file.
19	13		CHAR(*)	WDFFSCRS	Screen size table. This area defines the screen sizes valid for externally defined files. This is specified by the DSPSIZ keyword. When not specified, a default DSPSIZ(*DS3) is generated. Structure QDFFSCRA ("Screen Size Table (QDFFSCRA)") defines the entries. The elements are in the sequence that the DSPSIZ keywords are specified.

Screen Size Table (QDFFSCRA)

Screen ID array. The number of entries in this structure is defined by variable WDFFSCR in structure QDFFBASE. This structure is defined at variable WDFFSCRS in structure QDFFBASE. The structure is ARRAY(*).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(1)	WDFFSCIA	Screen ID. X'03' is defined as *DS3; X'04' is defined as *DS4.
1	1		CHAR(4)	*	Reserved.

Sort Sequence Table (QDFFSSEQ)

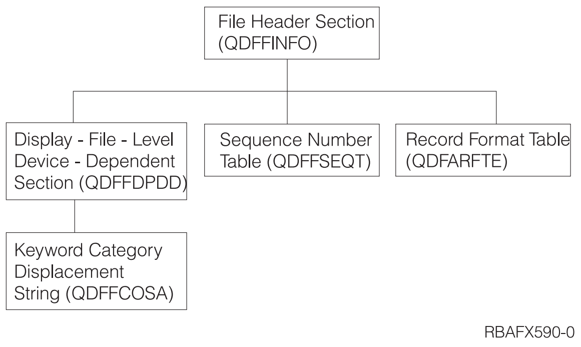
Sort sequence table information used for the ALTSEQ keyword. The displacement to this structure from the beginning of structure QDFFBASE is at variable WDFFSRSQ in QDFFBASE.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(256)	WDFFSST	Sort sequence table.
256	100		BIN(16)	WDFFSSC	CCSID associated with the table.
258	102		CHAR(10)	WDFFSSN	Table name.
268	10C		CHAR(10)	WDFFSSL	Library name.
278	116		CHAR(2)	WDFFSSFL	Indicator flags.
278	116	0	BIT(1)	WDFFSSUS	Weighted indicator. 0 is defined as shared weighted; 1 is defined as unique weighted.
278	116	1	BIT(1)	WDFFSSSB	Substitution characters indicator. 0 is defined as having no substitution characters; 1 is defined as having substitution characters.
278	116	2	BIT(14)	*	Reserved.
280	118		CHAR(26)	*	Reserved.

File Formats

File Header Section (page 138) shows the file section of the overview figure (DSPF0100 Format (page 135)).

File Header Section


File Header Section (QDFFINFO)
Display - File - Level Device - Dependent Section (QDFFDPDD)
Sequence Number Table (QDFFSEQT)
Record Format Table (QDFARFTE)
Keyword Category Displacement String (QDFFCOSA)
RBAFX590-0

File Header Section (QDFFINFO)

File header structure. The displacement to this structure from the beginning of structure QDFFBASE is at variable WDFFINOF in structure QDFFBASE.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(31)	WDFFDFLO	Length of the file header section. This includes the display-file-level device-dependent section. This is also the displacement from structure QDFFINFO to the record format table (see structure QDFARFTE, “Record Format Table (QDFARFTE)” on page 141).
4	4		BIN(31)	WDFFWUOF	Displacement to the where-used file-level information structure from structure QDFFINFO (see structure QDFWFLEI, “Where-Used File-Level Information Structure (QDFWFLEI)” on page 206).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
8	8		BIN(31)	WDFFFMWU	Length of file header section and the where-used file-level information structure.
12	C		BIN(31)	WDFFSEQO	Displacement from structure QDFFINFO to the sequence number table defined by structure QDFFSEQT (see "Sequence Number Table (QDFFSEQT)" on page 141). 0, if not present.
16	10		BIN(15)	WDFFSFL	Maximum number of entries in the selection tables defined by structure QDFFSTBL (see "Selection Table Entry (QDFFSTBL)" on page 158) at the record and field levels.
18	12		BIN(15)	WDFFSCE	Maximum number of entries in the selection tables for this file (structure QDFFSTBL, "Selection Table Entry (QDFFSTBL)" on page 158) at the record levels.
20	14		CHAR(2)	WDFFFFLG	File level flag.
20	14	0	BIT(1)	*	Reserved.
20	14	1	BIT(1)	WDFFGRPH	If on, the file contains at least one field with a graphic (G) data type.
20	14	2	BIT(14)	*	Reserved.
22	16		CHAR(12)	*	Reserved.
34	22		BIN(15)	WDFFXDOF	Displacement to display-file-level device-dependent section from structure QDFFINFO (see structure QDFFDPDD, "Display-File-Level Device-Dependent Section (QDFFDPDD)").

Display-File-Level Device-Dependent Section (QDFFDPDD)

Display device dependent section. The displacement to this structure from the beginning of structure QDFFINFO is at variable WDFFXDOF in QDFFINFO.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(8)	*	Reserved.
8	8		CHAR(2)	WDFFFKWD	Miscellaneous keyword indicator.
8	8	0	BIT(1)	WDFFOPEN	If on, OPENPRT keyword specified in file.
8	8	1	BIT(1)	WDFFCLRL	If on, CLRL keyword specified in some record format in this file.
8	8	2	BIT(1)	WDFFFICV	If on, IGCCNV keyword specified in file. **Note:** The IGCCNV keyword generates additional internal records (*IGCFMT).
8	8	3	BIT(1)	WDFFAGPH	If on, ALWGPH keyword specified on at least one record format in file.
8	8	4	BIT(1)	WDFFXHRD	If on, file-level HLPRCD keyword is specified.
8	8	5	BIT(1)	WDFFUDMT	If on, USRDSPMGT keyword is specified.
8	8	6	BIT(1)	WDFFPRPG	If on, PRINT(*PGM) keyword is specified.
8	8	7	BIT(1)	WDFFHSIO	If on, file-level HLPSCHIDX keyword is specified.
9	9	0	BIT(1)	WDFFXHTL	If on, file-level HLPTITLE keyword is specified.
9	9	1	BIT(1)	WDFFXUIM	If on, file-level HLPPNLGRP keyword is specified.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
9	9	2	BIT(1)	WDFFXHDC	If on, file-level HLPDOC keyword is specified.
9	9	3	BIT(1)	*	Reserved.
9	9	4	BIT(1)	WDFFALTN	If on, at least one ALTNAME keyword is specified in file.
9	9	5	BIT(1)	WDFFHFUL	If on, HLPFULL keyword is specified in file.
9	9	6	BIT(1)	WDFFESFL	If on, ERRSFL keyword is specified in file. **Note:** The ERRSFL keyword generates additional internal records (*ERRSFL).
9	9	7	BIT(1)	WDFFWDW	If on, WINDOW keyword is specified in file.
10	A		CHAR(2)	*	Reserved.
12	C		CHAR(1)	WDFFSHB1	Start-of-header (SOH) bits.
12	C	0	BIT(1)	WDFFSHCS	If on, CHECK(RLTB) keyword is specified.
12	C	1	BIT(1)	*	Reserved.
12	C	2	BIT(1)	WDFFAUTO	If on, DSPRL keyword is specified.
12	C	3	BIT(5)	*	Reserved.
13	D		CHAR(2)	*	Reserved.
15	F		CHAR(1)	WDFFSHRA	Row address of the message line for primary display size.
16	10		CHAR(1)	WDFFCKY1	File-level CA keys 17 through 24.
16	10	0	BIT(1)	WDFFCK24	If on, CA key 24 is specified.
16	10	1	BIT(1)	WDFFCK23	If on, CA key 23 is specified.
16	10	2	BIT(1)	WDFFCK22	If on, CA key 22 is specified.
16	10	3	BIT(1)	WDFFCK21	If on, CA key 21 is specified.
16	10	4	BIT(1)	WDFFCK20	If on, CA key 20 is specified.
16	10	5	BIT(1)	WDFFCK19	If on, CA key 19 is specified.
16	10	6	BIT(1)	WDFFCK18	If on, CA key 18 is specified.
16	10	7	BIT(1)	WDFFCK17	If on, CA key 17 is specified.
17	11		CHAR(1)	WDFFCKY2	File-level CA keys 9 through 16.
17	11	0	BIT(1)	WDFFCK16	If on, CA key 16 is specified.
17	11	1	BIT(1)	WDFFCK15	If on, CA key 15 is specified.
17	11	2	BIT(1)	WDFFCK14	If on, CA key 14 is specified.
17	11	3	BIT(1)	WDFFCK13	If on, CA key 13 is specified.
17	11	4	BIT(1)	WDFFCK12	If on, CA key 12 is specified.
17	11	5	BIT(1)	WDFFCK11	If on, CA key 11 is specified.
17	11	6	BIT(1)	WDFFCK10	If on, CA key 10 is specified.
17	11	7	BIT(1)	WDFFCK9	If on, CA key 9 is specified.
18	12		CHAR(1)	WDFFCKY3	File-level CA keys 1 through 8.
18	12	0	BIT(1)	WDFFCK8	If on, CA key 8 is specified.
18	12	1	BIT(1)	WDFFCK7	If on, CA key 7 is specified.
18	12	2	BIT(1)	WDFFCK6	If on, CA key 6 is specified.
18	12	3	BIT(1)	WDFFCK5	If on, CA key 5 is specified.
18	12	4	BIT(1)	WDFFCK4	If on, CA key 4 is specified.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
18	12	5	BIT(1)	WDFFCK3	If on, CA key 3 is specified.
18	12	6	BIT(1)	WDFFCK2	If on, CA key 2 is specified.
18	12	7	BIT(1)	WDFFCK1	If on, CA key 1 is specified.
19	13		CHAR(1)	WDFFMKWD	Miscellaneous keyword indicators.
19	13	0	BIT(1)	WDFFBRDR	If on, file-level WDWBORDER keyword is specified.
19	13	1	BIT(1)	*	Reserved.
19	13	2	BIT(1)	WDFFRTCR	If on, RTNCSRLOC keyword is specified.
19	13	3	BIT(1)	WDFFFFCP	If on, FLDCSRPRG keyword is specified.
19	13	4	BIT(1)	WDFFDSPP	If on, DSPATR program-to-system field is specified in file.
19	13	5	BIT(1)	WDFFHBKS	If on, HLPSHELF keyword is specified in file.
19	13	6	BIT(1)	WDFFINLYF	If on, CSRINPONLY keyword is specified in file.
19	13	7	BIT(1)	WDFFDBCSCNFLD	If on, CNTFLD keyword is used on a DBCS field in the file.
20	14		CHAR(1)	WDFFMKW2	More miscellaneous keywords.
20	14	0	BIT(1)	WDFFHTML	If on, the HTML keyword was specified in the file.
20	14	1	BIT(7)	*	Reserved.
21	15		CHAR(3)	*	Reserved.
24	18		BIN(15)	WDFFXDOC	Displacement to keyword category displacement string from structure QDFFINFO (see structure QDFFCOSA, "Keyword Category Displacement String (QDFFCOSA)" on page 159). 0, if no file keyword categories.

Record Format Table (QDFARFTE)

Record format table array. The number of entries in this structure is defined by variable WDFFRCS in structure QDFFBASE. The displacement to this structure from the beginning of structure QDFFINFO is at variable WDFFDFLO in QDFFINFO. The structure is ARRAY(*).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(10)	WDFARFNM	Record format name. Names that begin with * are internally generated record formats.
10	A		BIN(15)	WDFARCND	Miscellaneous record contents.
10	A	0	BIT(1)	WDFFRECD	If on, RECID keyword specified on this format.
10	A	1	BIT(15)	*	Reserved.
12	C		BIN(31)	WDFARFOF	Displacement to the record header section (see structure QDFFRINF, "Record Header Section (QDFFRINF)" on page 142) from structure QDFFINFO.

Sequence Number Table (QDFFSEQT)

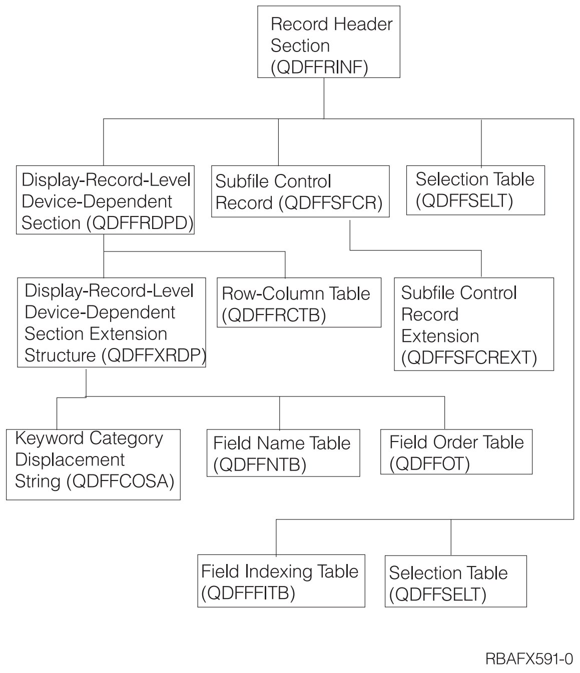
Sequence number table. The number of entries in this structure is defined by variable WDFFRCS in structure QDFFBASE. The displacement to this structure from the beginning of structure QDFFINFO is at variable WDFFSEQO in QDFFINFO. The structure is ARRAY(*).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(13)	WDFFSEQ	Level-check number for format. There is a one-to-one correspondence between this array and the entries in the record format table.
13	D		CHAR(3)	*	Reserved.

Record Formats

Record Header Section (page 142) shows the record section of the overview figure (DSPF0100 Format (page 135)).

Record Header Section


Record Header Section (QDFFRINF)
Display-Record-Level Device-Dependent Section (QDFFRDPD)
Subfile Control Record (QDFFSFCR)
Selection Table (QDFFSELT)
Display-Record-Level Device-Dependent Section Extension Structure (QDFFXRDP)
Row-Column Table (QDFFRCTB)
Subfile Control Record Extension (QDFFSFCREXT)
Keyword Category Displacement String (QDFFCOSA)
Field Name Table (QDFFNTB)
Field Order Table (QDFFOT)
Field Indexing Table (QDFFFITB)
Selection Table (QDFFSELT)
RBAFX591-0

Record Header Section (QDFFRINF)

Record header section. The displacement to this structure from the beginning of structure QDFFINFO is at variable WDFARFOF in structure QDFARFTE.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(31)	WDFFRDDO	Length of the record header section. This length includes the device-dependent sections (that is, it is the displacement to structure QDFFFINF for the first field in that record format).
4	4		BIN(31)	WDFFOFIT	The displacement from structure QDFFRINF to the field indexing table defined by structure QDFFFITB (see "Field Indexing Table (QDFFFITB)" on page 152).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
8	8		BIN(31)	WDFFSTBO	The displacement from structure QDFFRINF to the selection table defined by structure QDFFSELT (see "Selection Table (QDFFSELT)" on page 158). 0, if no selection table present.
12	C		BIN(31)	WDFFRFLG	Miscellaneous record contents.
12	C	0	BIT(1)	WDFFUDDS	If on, USRDFN keyword is specified.
12	C	1	BIT(1)	WDFFSFL	If on, SFL keyword is specified (next record is SFLCTL).
12	C	2	BIT(1)	WDFFSFLC	If on, SFLCTL keyword is specified (previous record is SFL).
12	C	3	BIT(1)	WDFFMSGR	If on, SFLMSGRCD keyword is specified.
12	C	4	BIT(1)	WDFFRICV	If on, IGCCNV record is specified. **Note:** The IGCCNV keyword generates additional internal records.
12	C	5	BIT(3)	*	Reserved.
13	D	0	BIT(1)	WDFFALLH	If on, all fields in format are hidden.
13	D	1	BIT(1)	*	Reserved.
13	D	2	BIT(1)	WDFFREXC	If on, DBCS data that can be processed is specified in record. This occurs when the O, J, or E data type is specified; when DBCS literals are specified on a DFT, DFTVAL, SFLMSG, RECID, ERRMSG, or RTGCON keyword.
13	D	3	BIT(1)	WDFFRIDV	If on, format requires a DBCS device.
13	D	4	BIT(1)	WDFFREXT	If on, extractable DBCS data is in format.
13	D	5	BIT(1)	WDFFRALT	If on, at least one field in format was specified as IGCALTTYP.
13	D	6	BIT(1)	WDFFMEMF	If on, CHECK(ME) or CHECK(MF) specified in at least one field in record.
13	D	7	BIT(1)	WDFFNDLC	If on, ALWENDLOC keyword is specified in record.
14	E	0	BIT(1)	WDFFRGPH	If on, graphic fields are specified in record.
14	E	1	BIT(1)	WDFFRCL	If on, RTNCSRLOC keyword is specified in record.
14	E	2	BIT(1)	WDFFMBAR	If on, MNUBAR keyword is specified in record.
14	E	3	BIT(1)	WDFFPULL	If on, PULLDOWN keyword is specified in record.
14	E	4	BIT(1)	WDFFPLSI	Selection indicators on PULLDOWN keyword. 0 is defined as *NOSLTIND; 1 is defined as *SLTIND (default).
14	E	5	BIT(1)	WDFFFCPF	If on, FLDCSRPRG specified on field in record.
14	E	6	BIT(1)	WDFFCNTMCFFLD	If on, CNTFLD, MLTCHCFLD, or SNGCHCFLD keyword is specified on a field within this record.
14	E	7	BIT(1)	WDFFEDTMSK	If on, EDTMSK keyword is specified in record.
15	F	0	BIT(1)	WDFFGRIDREC	If on, GRDRCD keyword is specified in record.
15	F	1	BIT(7)	*	Reserved.
16	10		BIN(15)	WDFFFLD	Number of fields in this record.
18	12		CHAR(4)	*	Reserved.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
22	16		BIN(15)	WDFFINDO	If INDARA keyword is specified and response indicators are in this record, this is the displacement from structure QDFFRINF to the response indicator keyword array (see structure QDFKMSCP, "Response Indicator Keyword Array (QDFKMSCP)" on page 167) in category 4. 0 means the INDARA keyword is not specified or if INDARA is specified, there are no response indicators.
24	18		CHAR(4)	*	Reserved.
28	1C		BIN(15)	WDFFRAOF	Displacements to display-record-level device-dependent section and subfile control record from structure QDFFRINF (see structures QDFFRDPD, "Display-Record-Level Device-Dependent Section (QDFFRDPD)," and QDFFSFCR, "Subfile Control Record (QDFFSFCR)" on page 148).

Display-Record-Level Device-Dependent Section (QDFFRDPD)

Display device-dependent section for nonsubfile records. Structure QDFFSFCR ("Subfile Control Record (QDFFSFCR)" on page 148) is used when subfiles are specified. The displacement to this structure from the beginning of structure QDFFRINF is an entry in the table at variable WDFFRAOF in QDFFRINF.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(31)	WDFFDRCO	Displacement to first row-column table (QDFFRCTB) from structure QDFFRINF. One row-column table exists for each display size (see variable WDFFSCR in structure QDFFBASE). The following may be used to access the correct table: let n = index into screen size array QDFFSCRA. (WDFFDRCO + (n - 1) * (WDFFFLD * LENGTH(WDFFRC)) + LENGTH(WDFFFRTO)) from QDFFRINF.
4	4		BIN(15)	WDFFINCP	Number of input-capable fields (that is, total input, both, and hidden).
6	6		BIN(15)	WDFFIBF	Number of input and both fields.
8	8		BIN(15)	WDFFOIS	Number of option indicators.
10	A		CHAR(2)	*	Reserved.
12	C		CHAR(4)	WDFACKYS	Indicates if a CA or CF key is specified. To determine which key (CA or CF) is specified, check the corresponding WDFFCKnn bit in structure QDFFDPDD (see "Display-File-Level Device-Dependent Section (QDFFDPDD)" on page 139).
12	C		CHAR(1)	WDFACKY1	Keys 1 through 8 without option indicators.
12	C	0	BIT(1)	WDFACK1	If on, CA/CF key 1 is specified.
12	C	1	BIT(1)	WDFACK2	If on, CA/CF key 2 is specified.
12	C	2	BIT(1)	WDFACK3	If on, CA/CF key 3 is specified.
12	C	3	BIT(1)	WDFACK4	If on, CA/CF key 4 is specified.
12	C	4	BIT(1)	WDFACK5	If on, CA/CF key 5 is specified.
12	C	5	BIT(1)	WDFACK6	If on, CA/CF key 6 is specified.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
12	C	6	BIT(1)	WDFACK7	If on, CA/CF key 7 is specified.
12	C	7	BIT(1)	WDFACK8	If on, CA/CF key 8 is specified.
13	D		CHAR(1)	WDFACKY2	Keys 9 through 16 without option indicators.
13	D	0	BIT(1)	WDFACK9	If on, CA/CF key 9 is specified.
13	D	1	BIT(1)	WDFACK10	if on, CA/CF key 10 is specified.
13	D	2	BIT(1)	WDFACK11	If on, CA/CF key 11 is specified.
13	D	3	BIT(1)	WDFACK12	If on, CA/CF key 12 is specified.
13	D	4	BIT(1)	WDFACK13	If on, CA/CF key 13 is specified.
13	D	5	BIT(1)	WDFACK14	If on, CA/CF key 14 is specified.
13	D	6	BIT(1)	WDFACK15	If on, CA/CF key 15 is specified.
13	D	7	BIT(1)	WDFACK16	If on, CA/CF key 16 is specified.
14	E		CHAR(1)	WDFACKY3	Keys 17 through 24 without option indicators.
14	E	0	BIT(1)	WDFACK17	If on, CA/CF key 17 is specified.
14	E	1	BIT(1)	WDFACK18	If on, CA/CF key 18 is specified.
14	E	2	BIT(1)	WDFACK19	If on, CA/CF key 19 is specified.
14	E	3	BIT(1)	WDFACK20	If on, CA/CF key 20 is specified.
14	E	4	BIT(1)	WDFACK21	If on, CA/CF key 21 is specified.
14	E	5	BIT(1)	WDFACK22	If on, CA/CF key 22 is specified.
14	E	6	BIT(1)	WDFACK23	If on, CA/CF key 23 is specified.
14	E	7	BIT(1)	WDFACK24	If on, CA/CF key 24 is specified.
15	F		CHAR(1)	WDFFCMDK	Other command keys without option indicators.
15	F	0	BIT(1)	WDFFRLUP	If on, ROLLUP keyword is specified.
15	F	1	BIT(1)	WDFFRLDN	If on, ROLLDOWN keyword is specified.
15	F	2	BIT(1)	WDFFPRNT	If on, PRINT keyword is specified.
15	F	3	BIT(1)	WDFFHOME	If on, HOME keyword is specified.
15	F	4	BIT(1)	WDFFCLR	If on, CLEAR keyword is specified.
15	F	5	BIT(1)	WDFFHELP	If on, HELP keyword is specified.
15	F	6	BIT(2)	*	Reserved.
16	10		CHAR(2)	WDFFPUTK	Miscellaneous PUT conditions.
16	10	0	BIT(1)	WDFFFSEL	If on, field selection.
16	10	1	BIT(1)	WDFFPUTR	If on, PUTRETAIN keyword is specified on some fields for this format.
16	10	2	BIT(1)	WDFFVSLN	If on, SLNO(*VAR) keyword is specified.
16	10	3	BIT(1)	WDFFALRL	If on, ALWROL keyword is specified.
16	10	4	BIT(1)	WDFFNOCO	Currently set for records containing floating point fields or DBCS data that requires a DBCS device (refer to WDFFRIDV).
16	10	5	BIT(1)	WDFFALGP	If on, unconditioned ALWGPH keyword is specified.
16	10	6	BIT(1)	WDFFRDMD	If on, DSPMOD keyword is specified.
16	10	7	BIT(1)	WDFFRMID	If on, MSGID keyword is specified on field in record.
17	11	0	BIT(1)	WDFFRKEY	If on, RETKEY keyword is specified.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
17	11	1	BIT(1)	WDFFRCKY	If on, RETCMDKEY keyword is specified.
17	11	2	BIT(1)	WDFFRDVL	If on, DFTVAL keyword is specified on field.
17	11	3	BIT(1)	WDFFVSL1	If on, SLNO(*VAR) keyword is specified and a field in row 1, column 1 is specified for at least one display size.
17	11	4	BIT(1)	WDFFMSGA	If on, unconditioned MSGALARM keyword is specified.
17	11	5	BIT(1)	WDFFRLST	If on, unconditioned RETLCKSTS keyword is specified.
17	11	6	BIT(1)	WDFFURDS	If on, unconditioned USRRSTDSP keyword is specified.
17	11	7	BIT(1)	WDFFRMVW	If on, unconditioned RMVWDW keyword is specified
18	12		CHAR(2)	WDFFGETK	Miscellaneous get conditions.
18	12	0	BIT(1)	*	Reserved.
18	12	1	BIT(1)	WDFFLOGN	If on, LOGINP keyword is specified.
18	12	2	BIT(1)	WDFFINZR	If on, INZRCD keyword is specified.
18	12	3	BIT(1)	WDFFRTND	If on, RTNDTA keyword is specified.
18	12	4	BIT(1)	WDFFUNLK	If on, UNLOCK keyword is specified.
18	12	5	BIT(1)	WDFFRSET	If on, UNLOCK(*MDTOFF) keyword specified or UNLOCK keyword specified with GETRETAIN.
18	12	6	BIT(1)	WDFFEARS	If on, UNLOCK(*ERASE) keyword specified or UNLOCK keyword specified without GETRETAIN.
18	12	7	BIT(1)	WDFFASUM	If on, ASSUME keyword is specified.
19	13	0	BIT(1)	WDFFKEEP	If on, KEEP keyword is specified.
19	13	1	BIT(1)	*	Reserved.
19	13	2	BIT(1)	WDFFWDWR	If on, WINDOW keyword specified in record.
19	13	3	BIT(1)	WDFFQILE	If on, SFLPGMQ(276) keyword is specified.
19	13	4	BIT(1)	WDFFSFLCHCCTL	If on, SFLCHCCTL keyword is specified.
19	13	5	BIT(3)	*	Reserved.
20	14		BIN(15)	WDFFERRM	Index to first field in index table with either ERRMSG or ERRMSGID keyword. 0, if record has no field with either keyword. See structure QDFFFITB, “Field Indexing Table (QDFFFITB)” on page 152.
22	16		CHAR(1)	WDFFBITS	Miscellaneous flags.
22	16	0	BIT(1)	WDFFERIN	If on, unconditioned ERASEINP(*MDTON) keyword is specified and ERASEINP(*ALL) is not specified.
22	16	1	BIT(1)	WDFFMDTO	If on, unconditioned MDTOFF(*UNPR) is specified and MDTOFF(*ALL) is not specified.
22	16	2	BIT(6)	*	Reserved.
23	17		CHAR(1)	WDFFBITF	Miscellaneous flags.
23	17	0	BIT(2)	*	Reserved.
23	17	2	BIT(2)	WDFFBLKC	Blink flags. X’00’ is defined as reserved; X’01’ is defined as blink cursor and keyword BLINK unconditioned; X’10’ is defined as reset blink cursor and no keyword BLINK; X’11’ is defined as reserved.
23	17	4	BIT(1)	WDFFNOLK	If on, no unconditioned lock. 0 is defined as lock unconditioned (do not unlock keyboard); 1 is defined as no LOCK keyword or conditioned lock (unlock keyboard).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
23	17	5	BIT(1)	WDFFALRM	If on, ALARM keyword unconditioned.
23	17	6	BIT(2)	*	Reserved.
24	18		BIN(15)	WDFFCGRI	Response indicator for record-level CHANGE keyword. For files with INDARA keyword, this is the response indicator minus 1. For those without INDARA keyword, this is the response indicator input buffer displacement. -1 shows keyword is not present.
26	1A		CHAR(2)	WDFFHFLG	Help flags.
26	1A	0	BIT(1)	WDFFHSEQ	If on, HLPSEQ keyword on record.
26	1A	1	BIT(1)	WDFFHLP	If on, help specifications on record.
26	1A	2	BIT(1)	WDFFNHLP	If on, record cannot be used as help text. It contains one of the keywords USRDFN, SFL, or SFLCTL.
26	1A	3	BIT(1)	WDFFHRTN	If on, HLPRTN keyword on record.
26	1A	4	BIT(1)	WDFFHTLE	If on, HLPTITLE keyword on record.
26	1A	5	BIT(1)	WDFFHCLR	If on, HLPCLR keyword on record.
26	1A	6	BIT(1)	WDFFCHNG	If on, no parameter for CHANGE keyword.
26	1A	7	BIT(1)	WDFFRPGM	If on, PRINT keyword on record level with *PGM.
27	1B	0	BIT(1)	WDFFHLPC	If on, HLPCMDKEY keyword on record.
27	1B	1	BIT(1)	WDFFRSTCSR	If on, *RSTCSR parameter is specified on the PULLDOWN keyword on the record.
27	1B	2	BIT(1)	WDFFINLY	If on, CSRINPONLY keyword is specified and is unoptioned.
27	1B	3	BIT(1)	WDFFNOSEP	If on, *NOSEPARATOR parameter is specified on the MNUBAR keyword on this record.
27	1B	4	BIT(4)	*	Reserved.
28	1C		BIN(15)	WDFFXRDO	Displacement to display-record-level device-dependent extension structure from structure QDFFRINF (see structure QDFFXRDP, “Display-Record-Level Device-Dependent Section Extension Structure (QDFFXRDP)”).
30	1E		CHAR(2)	*	Reserved.
32	20		BIN(15)	WDFFRDOC	Displacement to keyword category displacement string from structure QDFFRINF. (See structure QDFFCOSA, “Keyword Category Displacement String (QDFFCOSA)” on page 159.) 0, if no keyword categories.

Display-Record-Level Device-Dependent Section Extension Structure (QDFFXRDP)

Extension structure. The displacement to this structure from the beginning of structure QDFFRINF is at variable WDFFXRDO in structure QDFFRDPD.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(31)	WDFFOTO	Displacement to field order table from structure QDFFRINF (see structure QDFFOT, “Field Order Table (QDFFOT)” on page 152).
4	4		BIN(31)	WDFFNRCO	Displacement to first field name in row-column order table (see variable WDFFDRCO in structure QDFFRDPD, (page 144). For every row-column table, there is a corresponding field name in row-column order in the field name table (see structure QDFFNTB, “Field Name Table (QDFFNTB)” on page 152).
8	8		CHAR(4)	*	Reserved.
12	C		BIN(15)	WDFFNUMOFSEGS	Number of segments in record for CNTFLD and EDTMSK.
14	E		CHAR(2)	*	Reserved.
16	10		BIN(15)	WDFFSFLCHCTLO	Buffer displacement to the field containing control for selection list.
18	12		CHAR(6)	*	Reserved.

Subfile Control Record (QDFFSFCR)

Display device-dependent section for records specifying subfiles. This structure replaces structure QDFFRDPD when subfiles are specified (variable WDFFSFLC in structure QDFFRINF is set on. The displacement to this structure from the beginning of structure QDFFRINF is an entry in the table at variable WDFFRAOF in QDFFRINF.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(34)	WDFFSFLG	Display-record-level device-dependent section (structure QDFFRDPD, “Display-Record-Level Device-Dependent Section (QDFFRDPD)” on page 144) is mapped here.
34	22		CHAR(1)	WDFFSFEN	Command key associated with SFLENTER keyword. X’00’ indicates the keyword is not present.
35	23		CHAR(1)	WDFFSFDR	Command key associated with SFLDROP or SFLFOLD keyword. X’00’ indicates neither keyword is specified (see WDFFSFFD in this table).
36	24		CHAR(1)	WDFFSFLFLG	Subfile flags.
36	24	0	BIT(1)	WDFFSFLSNGCHC	If on, SFLSNGCHC keyword is specified.
36	24	1	BIT(1)	WDFFSFLMLTCHC	If on, SFLMLTCHC keyword is specified.
36	24	2	BIT(1)	WDFFSFLSELRSC	If on, *RSTCSR parameter is specified on SFLMLTCHC or SFLSNGCHC keyword.
36	24	3	BIT(1)	WDFFSFLSELSND	If on, *SLTIND parameter is specified on SFLMLTCHC or SFLSNGCHC keyword.
36	24	4	BIT(1)	WDFFSFLSELAST	If on, *AUTOSLT parameter is specified on SFLSNGCHC keyword.
36	24	5	BIT(1)	WDFFSFLSCRBAR	If on, SFLEND(*SCRBAR) keyword is specified.
36	24	6	BIT(1)	WDFFSFLRTNSEL	If on, SFLRTNSEL keyword is specified.
36	24	7	BIT(1)	WDFFSFLSCROLL	If on, SFLSCROLL keyword is specified.
37	25		CHAR(1)	WDFFSFST	Miscellaneous flags.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
37	25	0	BIT(1)	WDFFSFNA	If on, SFLRNA keyword is specified.
37	25	1	BIT(1)	WDFFSFCU	If on, SFLRCDNBR(CURSOR) keyword is specified.
37	25	2	BIT(1)	WDFFSFDM	If on, DSPMOD keyword is specified.
37	25	3	BIT(1)	WDFFSFFD	0 indicates the initial display is fold; 1 indicates the initial display is drop. If WDFFSFDR equals X'00', there is no SFLDROP or SFLFOLD keyword, and this value equals 0. If WDFFSFDR does not equal X'00', either this value equals 0 (SFLFOLD) or 1 (SFLDROP). **Note:** Also refer to comments for variable WDFFSFDR in this structure.
37	25	4	BIT(1)	WDFFSFFDI	0 indicates that SFLDROP or SFLFOLD are not optioned; use WDFFSFFD to determine which one to use. 1 indicates that SFLDROP and SFLFOLD are optioned; use indicators. **Note:** Also refer to comments for variable WDFFSFDR in this structure.
37	25	5	BIT(1)	WDFFSFEM	If on, SFLEND(*MORE) keyword is specified.
37	25	6	BIT(1)	WDFFSFLRCDtop	If on, SFLRCDNBR(*top) keyword is specified.
37	25	7	BIT(1)	WDFFSFLSELSTE	If on, *AUTOSLTENH parameter is specified on SFLSNGCHC keyword.
38	26		BIN(15)	WDFFSFPQ	Contains the value specified for the SFLPGMQ keyword.
40	28		BIN(15)	WDFFSFVL	SFLROLVAL field length. 0 indicates that the keyword is not specified.
42	2A		BIN(15)	WDFFSFVO	Displacement in input buffer to SFLROLVAL.
44	2C		BIN(15)	WDFFSFFI	Index into field indexing table of field with SFLROLVAL.
46	2E		BIN(15)	WDFFSFL	SFLRCDNBR field length. 0 indicates that the keyword is not specified.
48	30		BIN(15)	WDFFSFO	Displacement in output buffer to SFLRCDNBR.
50	32		BIN(15)	WDFFSFLEXTOFF	Displacement to the QDFFSFCREXT extension structure (see "Subfile Control Record Extension (QDFFSFCREXT)" on page 150) from this structure.
52	34		CHAR(1)	WDFFSFLNOFL	Miscellaneous bits.
52	34		CHAR(1)	WDFFSFNOFL	Miscellaneous flags.
52	34	0	BIT(1)	WDFFSFLSELNRS	If on, *NORSTCSR parameter is specified on SFLMLTCHC or SFLSNGCHC keyword.
52	34	1	BIT(1)	WDFFSFLSELNST	If on, *NOAUTOSLT parameter is specified on SFLSNGCHC keyword.
52	34	2	BIT(6)	*	Reserved
53	35		CHAR(1)	*	Reserved.
54	36		CHAR(*)	WDFFSFPM	SFL parameter values (see structure QDFFSFHR, "Subfile Control Entry (QDFFSFHR)" on page 150). One entry is present for each specified display size (see WDFFSCRS in structure QDFFBASE). The order of this array is the same as structure QDFFSCRA ("Screen Size Table (QDFFSCRA)" on page 137).

Subfile Control Entry (QDFFSFHR)

Subfile control entry in the subfile control record. This structure is defined at variable WDFFSFPM in structure QDFFSFCR. The structure is ARRAY(*).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFFSFSZ	SFLSIZ.
2	2		BIN(15)	WDFFSFPG	SFLPAG. If this is a field selection subfile, this is the number of lines occupied by subfile. If this is a nonfield selection subfile, this is the maximum number of subfile records on the screen.
4	4		CHAR(2)	*	Reserved.
6	6		BIN(15)	WDFFSFT	Number of fields not dropped, that is, the number of fields on first line of SFL record with SFLDROP specified.
8	8		BIN(15)	WDFFSFR1	Subfile start row.
8	8		CHAR(1)	*	Reserved.
9	9		CHAR(1)	WDFFSFSR	Subfile start row. For SFLMSGRCD, this is line number.
10	A		BIN(15)	WDFFSFR2	Subfile end row.
10	A		CHAR(1)	*	Reserved.
11	B		CHAR(1)	WDFFSFER	Subfile end row.
12	C		CHAR(4)	WDFFSFLN	Horizontal subfile (SFLLIN). 0 is defined as not horizontal subfile.
12	C		BIN(15)	WDFFSFH1	Number of horizontal records per line.
14	E		BIN(15)	WDFFSFH2	Number of characters from field 1, record *n* to field 1, record *n*+1.
16	10		BIN(15)	WDFFSFF	Number of fields per record.
18	12		CHAR(6)	*	Reserved.

Subfile Control Record Extension (QDFFSFCREXT)

Subfile control record extension entry in the subfile control record (see structure QDFFSFCR, “Subfile Control Record (QDFFSFCR)” on page 148). Variable WDFFSFLEXTOFF contains the displacement to this structure from structure QDFFSFCR.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFFSFLSCRLLO	Displacement to the field with the SFLSCROLL keyword.
2	2		BIN(15)	WDFFSFLSIZSFO	Displacement to the field specified on the SFLSIZ keyword. -1 indicates a number was specified.
4	4		BIN(15)	WDFFSFLSELOFF	Displacement to the field specified on the SFLMLTCHC keyword that is used to tell the application the number of selections made from the selection list.
6	6		CHAR(1)	WDFFSFLSELCH1	Primary character to be used to indicate a selection list item has been selected.
7	7		CHAR(1)	WDFFSFLSELCH2	Secondary character to be used to indicate a selection list item has been selected.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
8	8		CHAR(8)	*	Reserved.

Row-Column Table (QDFFRCTB)

Row-column table, one table per screen size. The displacement to this structure from the beginning of structure QDFFRINF is at variable WDFFDRCO (page 144) in structure QDFFRDPD.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(1)	WDFFFRRO	From row of attribute of first field in format.
1	1		CHAR(1)	WDFFFRCO	From column of attribute of first field in format.
2	2		CHAR(1)	WDFFTORO	To row of last data character of last field in format (excluding trailing attribute).
3	3		CHAR(1)	WDFFTOCO	To column of last data character of last field in format (excluding trailing attribute).
4	4		CHAR(1)	WDFFRBIT	Miscellaneous flags.
4	4	0	BIT(1)	WDFFMDF	Multiple defined fields (MDF) present for this screen size. MDF fields are defined to be a group of fields that have the same beginning row-column, and the first field in the group must have field selection.
4	4	1	BIT(1)	WDFFFRC1	First field in the record has attribute in column 1 for this screen size.
4	4	2	BIT(1)	WDFFTRAT	If on, the trailing attribute for this screen size was in column one.
4	4	3	BIT(1)	WDFFR1C1	First field in record begins in row 1, column 1 for this screen size.
4	4	4	BIT(1)	WDFFR2C1	First field in record begins in row 2, column 1 for this screen size and the SLNO(nn) keyword.
4	4	5	BIT(3)	*	Reserved.
5	5		CHAR(1)	*	Reserved.
6	6		CHAR(*)	WDFFRC	Row-column table, one entry per field (see structure QDFFRCTE, “Row-Column Table Entry (QDFFRCTE)”).

Row-Column Table Entry (QDFFRCTE)

Row-column table with one table entry per field. The number of entries in this structure is defined by variable WDFFFLD in structure QDFFRINF. This structure is defined at variable WDFFRC in structure QDFFRCTB. The structure is ARRAY(*).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(1)	WDFFSROW	Starting row. X’FF’ indicates that the location for the secondary display size was *NOLOC, or was a hidden field, a program field, or a message line.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
1	1		CHAR(1)	WDFFSCOL	Starting column. X'FF' indicates that the location for the secondary display size was *NOLOC, or was a hidden field, a program field, or a message line.

Field Name Table (QDFFNTB)

Field name table with one field name entry per field. This structure is present when the RTNCSRLOC keyword is specified in the DDS. The number of entries in this structure is defined by variable WDFFFLD in structure QDFFRINF. The displacement to this structure from the beginning of structure QDFFRINF is at variable WDFFNRCO in structure QDFFXRDP. The structure is ARRAY(*).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(10)	WDFFNAM	Field name entry.

Field Order Table (QDFFOT)

Field order table with one field order entry per field. This structure is present when the USRDFNMGT keyword is specified in the DDS. The number of entries in this structure is defined by variable WDFFFLD in structure QDFFRINF. The displacement to this structure from the beginning of structure QDFFRINF is at variable WDFFOTO (page 147) in structure QDFFXRDP. The structure is ARRAY(*).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFFLD	Order of field in DDS source.

Field Indexing Table (QDFFFITB)

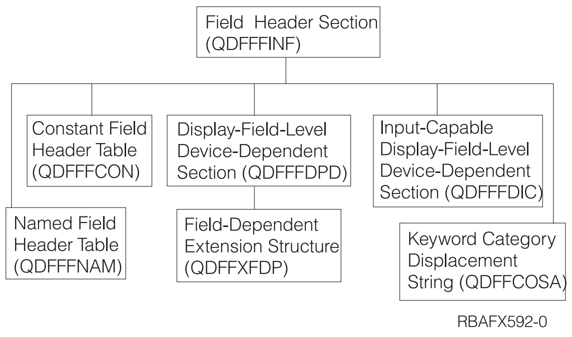
Field indexing table. The number of entries in this structure is defined by variable WDFFFLD. The displacement to this structure from the beginning of structure QDFFRINF is at variable WDFFOFIT in QDFFRINF. The structure is ARRAY(*).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(31)	WDFFFOFS	Displacement from the record header (structure QDFFRINF) to this field's header section (see structure QDFFFINF, "Field Header Section (QDFFFINF)" on page 153).
4	4		BIN(15)	WDFFSELI	Index to the entry in the selection table (see variable WDFFSTE in structure QDFFSELT) for the condition selecting this field. 1 represents no field selection.
6	6		BIN(15)	WDFFDLEN	Display length. Edited field length and UCS-2 displayed field length. For floating point edited fields, this value is the significand plus 7. For nonfloating-point edited fields when the FLTFIXDEC keyword is specified, this value is the length specified for the field plus 2. When the FLTFIXDEC keyword is not specified, this value is 7 plus the length specified for the field.

Field Formats

Field Header Section (page 153) shows the field section of the overview figure (DSPF0100 Format (page 135)).

Field Header Section


Field Header Section (QDFFFINF)
Constant Field Header Table (QDFFFCON)
Display-Field-Level Device-Dependent Section (QDFFFDPD)
Input-Capable Display-Field-Level Device-Dependent Section (QDFFFDIC)
Named Field Header Table (QDFFFNAM)
Field-Dependent Extension Structure (QDFFXFDP)
Keyword Category Displacement String (QDFFCOSA)
RBAFX592-0

Field Header Section (QDFFFINF)

Field header declare. The displacement to this structure from the beginning of structure QDFFRINF is at variable WDFFFOFS in structure QDFFFITB.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		BIN(15)	WDFFFLEN	Length of this declare, including all the device-dependent sections.
2	2		CHAR(1)	WDFFFIOA	Field attribute. X'01' indicates Constant (see structure QDFFFCON, "Constant Field Header Table (QDFFFCON)" on page 154), X'02' indicates Output (O), X'03' indicates Message (M), X'04' indicates Input (I), X'05' indicates Both (B), X'06' indicates Hidden (H), and X'07' indicates Program to System (P).
3	3		CHAR(1)	WDFFBFLG	Miscellaneous flags.
3	3	0	BIT(1)	WDFFDATE	If on, DATE keyword is specified.
3	3	1	BIT(1)	WDFFDATY	If on, DATEY keyword is specified.
3	3	2	BIT(1)	WDFFTIME	If on, TIME keyword is specified.
3	3	3	BIT(1)	WDFFFOLD	If on, BLKFOLD keyword is specified.
3	3	4	BIT(1)	WDFFEDIT	If on, EDTCDE or EDTWRD keyword is specified.
3	3	5	BIT(1)	WDFFINBT	If on, field is either input or both.
3	3	6	BIT(1)	WDFFDFT	If on, DFT or DFTVAL keyword is specified.
3	3	7	BIT(1)	WDFFFALT	If on, IGCALTTYP keyword is specified.
4	4		CHAR(1)	WDFFFBIT	Miscellaneous flags.
4	4	0	BIT(1)	WDFFIGCC	If on, DBCS literals are specified on DFT or DFTVAL keyword.
4	4	1	BIT(1)	WDFFFCSO	If on, first character of DFT or DFTVAL keyword is shift out (SO).
4	4	2	BIT(1)	WDFFOPDV	If on, optioned DFTVAL keyword is specified.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
4	4	3	BIT(1)	WDFFALWE	If on, ALWENDLOC applies to field. Trailing attribute byte should be truncated for at least one display size.
4	4	4	BIT(1)	WDFFUSER	If on, USER keyword is specified.
4	4	5	BIT(1)	WDFFSYSN	If on, SYSNAME keyword is specified.
4	4	6	BIT(1)	WDFFEDFT	If on, EDTWRD was generated due to the DATE or TIME keyword, or due to the L, T, or Z edit code.
4	4	7	BIT(1)	WDFF_EDTCDE_Y	If on, the edit code specified on the EDTCDE keyword is used for formatting dates. The edit code is either a W or a Y.
5	5		CHAR(1)	*	Reserved.
6	6		CHAR(*)	WDFFFTBE	Field header table entries. Use structure QDFFFCON (“Constant Field Header Table (QDFFFCON)”) for constant fields and structure QDFFFNAM (“Named Field Header Table (QDFFFNAM)”) for named fields.

Constant Field Header Table (QDFFFCON)

Field header declare for constant fields. This structure is defined at variable WDFFFTBE in structure QDFFFINF.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(2)	*	Reserved.
2	2		BIN(15)	WDFFFAOC	Displacement for constant fields to field-level device-dependent sections (structure QDFFFDPD, “Display-Field-Level Device-Dependent Section (QDFFFDPD)” on page 155) from structure QDFFFINF (“Field Header Section (QDFFFINF)” on page 153).

Named Field Header Table (QDFFFNAM)

Field header declare for named fields. This structure is defined at variable WDFFFTBE in structure QDFFFINF.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFFINPO	Input buffer displacement. -1 indicates no buffer location.
2	2		BIN(15)	WDFFOUTO	Output buffer displacement. -1 indicates no buffer location.
4	4		BIN(15)	WDFFPLEN	Program length. User’s program field length for floating point fields indicates precision 4 is defined as *SINGLE; 8 is defined as *DOUBLE.
6	6		CHAR(1)	WDFFDEC	Decimals (X’00’ through X’1F’). X’FF’ indicates field is character or DBCS-capable.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
7	7		CHAR(1)	WDFFKBDT	Keyboard shift and data type. X’00’ indicates Alpha shift/character (A), X’01’ indicates Alpha only (X), X’02’ indicates Numeric shift (N), X’03’ indicates Numeric only (Y), X’04’ indicates Katakana (K), X’05’ indicates Digits only (D), X’06’ indicates Inhibit keyboard (I), X’07’ indicates Signed numeric/zoned (S), X’08’ indicates Binary (B), X’09’ indicates Packed (P), X’0A’ indicates Floating (F), X’0B’ indicates DBCS (J), X’0C’ indicates Open (O), X’0D’ indicates Either (E), X’0E’ indicates Numeric-only character (M), X’0F’ indicates Graphic (G), X’10’ indicates Date (L), X’11’ indicates Time (T), and X’12’ indicates Timestamp (Z).
8	8		CHAR(2)	*	Reserved.
10	A		BIN(15)	WDFFFAOF	Displacement for nonconstant (named) fields to display-field-level device-dependent section (structure QDFFFDPD, “Display-Field-Level Device-Dependent Section (QDFFFDPD)”) from structure QDFFFINF (“Field Header Section (QDFFFINF)” on page 153).

Display-Field-Level Device-Dependent Section (QDFFFDPD)

Display device-dependent section. The displacement to this structure from the beginning of structure QDFFFINF is an entry in the table at variable WDFFFAOF in structure QDFFFNAM.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFFFLGS	Miscellaneous flags.
0	0	0	BIT(1)	WDFFDSPC	If on, unconditioned DSPATR(PC) keyword is specified.
0	0	1	BIT(1)	WDFFUCND	If on, unconditioned DSPATR(ND) keyword is specified.
0	0	2	BIT(1)	WDFFFXDC	If on, FLTFIXDEC keyword is specified.
0	0	3	BIT(1)	WDFFIACV	If on, IGCANKCNV keyword is specified.
0	0	4	BIT(1)	WDFFCSCP	If on, CHRID keyword is specified.
0	0	5	BIT(1)	WDFFMGID	If on, MSGID keyword is specified.
0	0	6	BIT(1)	WDFFDPNR	If on, DUP keyword is specified without a response indicator on a numeric field.
0	0	7	BIT(1)	WDFFDSPN	Field’s base cursor position. If on, the field is input-capable and no unoptioned DSPATR(PR) or no unoptioned DSPATR(PC) is in any field in the record.
1	1		CHAR(1)	WDFFSA	Default screen attribute byte for workstation.
1	1	0	BIT(3)	*	Reserved. Always B’001’.
1	1	3	BIT(1)	WDFFCLOS	If on, unconditioned DSPATR(CS) keyword is specified.
1	1	4	BIT(1)	WDFFBLNK	If on, unconditioned DSPATR(BL) keyword is specified.
Note: If the following three bits are on, unconditioned DSPATR(ND) is specified.					
1	1	5	BIT(1)	WDFFUDLN	If on, unconditioned DSPATR(UL) keyword is specified.
1	1	6	BIT(1)	WDFFHILI	If on, unconditioned DSPATR(HI) keyword is specified.
1	1	7	BIT(1)	WDFFRVIM	If on, unconditioned DSPATR(RI) keyword is specified.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
2	2		BIN(15)	WDFFXFDO	Displacement to field-dependent extension structure from QDFFFINF (see structure QDFFXFDP, “Field-Dependent Extension Structure (QDFFXFDP)” on page 157). 0 indicates no extension structure is present.
4	4		BIN(15)	WDFFFDOC	Displacement to keyword category displacement string from structure QDFFFINF (see structure QDFFCOSA, “Keyword Category Displacement String (QDFFCOSA)” on page 159). 0, if no keyword categories.
6	6		CHAR(*)	WDFFFICE	Input-capable display field-level device-dependent section entries (see structure QDFFFDIC, “Input-Capable Display Field-Level Device-Dependent Section (QDFFFDIC)”). Only used for types X’04’ (input) and X’05’ (both); see variable WDFFFIOA in structure QDFFFINF.

Input-Capable Display Field-Level Device-Dependent Section (QDFFFDIC)

Input-capable display device-dependent section. This structure is used for types X’04’ (input) and X’05’ (both); see variable WDFFFIOA in structure QDFFFINF. This structure is defined at variable WDFFFICE) in structure QDFFFDPD.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(2)	WDFFFWFW	Miscellaneous flags.
0	0	0	BIT(2)	*	Reserved.
0	0	2	BIT(1)	WDFFFWPR	If on, unconditioned DSPATR(PR) keyword is specified.
0	0	3	BIT(1)	WDFFFWDP	If on, unconditioned DUP keyword is specified.
0	0	4	BIT(1)	WDFFFWMD	If on, unconditioned DSPATR(MDT) keyword is specified.
0	0	5	BIT(3)	WDFFFWSF	Keyboard shift. B’000’ indicates alpha shift, B’001’ indicates alpha only, B’010’ indicates numeric shift (also floating point), B’011’ indicates numeric only (also numeric-only character keyboard shift), B’100’ indicates Katakana/CHECK(RL), B’101’ indicates digits only, B’110’ indicates inhibit keyboard, B’111’ indicates signed numeric.
1	1	0	BIT(1)	WDFFFWRA	If on, unconditioned AUTO(RA) keyword is specified.
1	1	1	BIT(1)	WDFFFWFE	If on, CHECK(FE) keyword is specified.
1	1	2	BIT(1)	WDFFFWLW	Lowercase (not monocase). 0 indicates lowercase; 1 indicates not lowercase (uppercase).
1	1	3	BIT(1)	*	Reserved.
1	1	4	BIT(1)	WDFFFWME	If on, unconditioned CHECK(ME) keyword is specified.
1	1	5	BIT(3)	WDFFFWAJ	Adjustments. B’000’ indicates no adjustment, B’101’ indicates AUTO(RAZ), B’110’ indicates AUTO(RAB), B’111’ indicates CHECK(MF).
2	2		CHAR(1)	WDFFSSKW	Keywords present.
2	2	0	BIT(1)	WDFFBLKS	If on, BLANKS keyword is specified.
2	2	1	BIT(1)	WDFFSSCH	If on, CHANGE keyword is specified.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
2	2	2	BIT(1)	WDFFSSDR	If on, DUP keyword is specified with a response indicator.
2	2	3	BIT(1)	WDFFSSDP	If on, DUP keyword is specified with or without a response indicator.
2	2	4	BIT(1)	WDFFSSAB	If on, CHECK(AB) keyword is specified.
2	2	5	BIT(1)	WDFFDSOD	If on, DSPATR(OID) keyword is specified.
2	2	6	BIT(1)	WDFFDSSP	If on, DSPATR(SP) keyword is specified.
2	2	7	BIT(1)	WDFFVLCK	If on, validity checking keywords specified in category 25, "Category 25 (GET Validation Keywords)" on page 190 (that is, category 25 is present).
3	3		CHAR(1)	WDFFCHKB	Miscellaneous flags.
3	3	0	BIT(1)	WDFFCM10	If on, CHECK(M10) keyword is specified.
3	3	1	BIT(1)	WDFFCM11	If on, CHECK(M11) keyword is specified.
3	3	2	BIT(1)	WDFFM10F	If on, CHECK(M10F) keyword is specified.
3	3	3	BIT(1)	WDFFM11F	If on, CHECK(M11F) keyword is specified.
3	3	4	BIT(4)	*	Reserved.

Field-Dependent Extension Structure (QDFFXFDP)

Field-dependent extension structure. The displacement to this structure from the beginning of structure QDFFFINF is at variable WDFFXFDO in structure QDFFFDPD.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(2)	WDFFKFLG	Miscellaneous flag.
0	0	0	BIT(1)	WDFFNOBA	If on, field has no beginning attribute.
0	0	1	BIT(1)	WDFFNOEA	If on, field has no ending attribute.
0	0	2	BIT(1)	*	Reserved.
0	0	3	BIT(1)	WDFFFDCP	If on, this field is referenced by another field using the FLDCSRPRG keyword.
0	0	4	BIT(1)	WDFFSFCP	If on, SFLCSRPRG keyword specified on field.
0	0	5	BIT(1)	WDFFMLTC	If on, MLTCHCFLD keyword is specified.
0	0	6	BIT(1)	WDFFSNGC	If on, SNGCHCFLD or PSHBTNFLD keyword is specified.
0	0	7	BIT(1)	WDFFCNTF	If on, CNTFLD keyword is specified.
1	1	0	BIT(1)	WDFFENFA	If on, ENTFLDATR keyword is specified.
1	1	1	BIT(1)	WDFFFCRP	If on, FLDCSRPRG keyword is specified.
1	1	2	BIT(1)	WDFFEDTM	If on, EDTMSK keyword is specified.
1	1	3	BIT(1)	WDFFPFLD	If on, field has associated program-to-system field.
1	1	4	BIT(1)	WDFFNOCC	If on, NOCCSID keyword is specified.
1	1	5	BIT(1)	WDFFPUSHBTN	If on, PSHBTNFLD keyword is specified.
1	1	6	BIT(1)	WDFFCHCHDHEXP	If on, structure QDFKCHC ("CHCFLD Keyword Structure (QDFKCHC)" on page 193) has an extension structure appended to it.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
1	1	7	BIT(1)	WDFFWRDWRAP	If on, WRDWRAP keyword is specified.
2	2		BIN(15)	WDFFFLDINX	Field index of current field.
4	4		CHAR(1)	WDFFXLFLGS	Miscellaneous flags.
4	4	0	BIT(1)	WDFFVALNUM	If on, VALNUM keyword is specified.
4	4	1	BIT(1)	WDFFUCS2OF	If on, WDFF_UCS2_CCSID contains the output buffer offset where the CCSID is located.
4	4	2	BIT(6)	*	Reserved.
5	5		BIN(16)	WDFF_UCS2_ CCSID	The UCS-2 CCSID specified on the CCSID keyword. If WDFFUCS2OF is on, this is the output buffer offset where the CCSID is located.
7	7		CHAR(1)	*	Reserved.

Selection Table (QDFFSELT)

Selection table. The table entries are defined in structure QDFFSTBL (page “Selection Table Entry (QDFFSTBL)”). The entries in the where-used section (“Where-Used Formats” on page 206) are stored in the same order as the selection table. The displacement to this structure from the beginning of structure QDFFRINF is at variable WDFFSTBO in structure QDFFRINF.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(31)	WDFFSTLN	Selection table length.
4	4		BIN(15)	WDFFSTT	Total number of table entries used by the display to resolve record- and field-level selection entries.
6	6		CHAR(2)	*	Reserved.
8	8		CHAR(*)	WDFFSTE	Selection table entries (see structure QDFFSTBL, “Selection Table Entry (QDFFSTBL)”).

Selection Table Entry (QDFFSTBL)

Selection table entry. The number of entries in this structure is defined by variable WDFFSTT in structure QDFFSELT. This structure is defined at variable WDFFSTE in structure QDFFSELT. The structure is ARRAY(*).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFFSTEL	Length of this table entry.
2	2		BIN(15)	WDFFCND	Number of conditions in the entry.
4	4		ARRAY(*) OF CHAR(1)	WDFFSELM	Array of selection table indicators. The number of entries in this structure is defined by variable WDFFCND in this table. Each character contains a displacement into the output buffer for an option indicator. An entry is used to designate whether the indicator must be on (X’F1’) or off (X’F0’). If an entry is on, the indicator must be on; if it is off, the indicator must be off. The value X’7F’ in this field designates the end of the entry.

Keyword Category Displacement String (QDFFCOSA)

Category displacement string. This structure occurs for each display file-, record-, or field-level section that has keyword structures. For file-level sections, the displacement to this structure is from the beginning of structure QDFFINFO at variable WDFFXDOC) in structure QDFFDPDD. For record-level sections, the displacement to this structure is from the beginning of structure QDFFRINF at variable WDFFRDOC) in structure QDFFRDPD. For field-level sections, the displacement to this structure is from the beginning of structure QDFFFINF at variable WDFFFDOC) in structure QDFFFDPD.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFFCCT	Number of entries in the category displacement string.
2	2		CHAR(*)	WDFFCCOS	Category displacement string (see structure QDFFCCOA, "Keyword Category Displacement String Entry (QDFFCCOA)").

Keyword Category Displacement String Entry (QDFFCCOA)

Category displacement string array. Each keyword category type that is present in the file, record, or field section has an entry. The number of entries in this structure is defined by variable WDFFCCT in structure QDFFCOSA. This structure is defined at variable WDFFCCOS in structure QDFFCOSA. The structure is ARRAY(*).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFFCAID	Category ID (X'01' through X'FF').
1	1		BIN(15)	WDFFCAOF	Displacement to category from the start of each section header (see structure QDFFINFO on "File Header Section (QDFFINFO)" on page 138 for file-level keywords, structure QDFFRINF on "Record Header Section (QDFFRINF)" on page 142 for record-level keywords, or structure QDFFFINF on "Field Header Section (QDFFFINF)" on page 153 for field-level keywords).

Keyword Formats

Category 1 (File-Level Keywords)

The following table shows the keyword ID that corresponds to the file-level keywords. Not all keywords require a structure. There are no structures for keyword IDs X'01', X'03', and X'0D'. The text associated with the HLPTITLE keyword is contained in variable WDFKFLNM in structure QDFKFLPP.

ID	Keyword	ID	Keyword
X'01'	PASSRCD	X'07'	HLPDOC
X'02'	MSGLOC	X'08'	HLPSCHIDX
X'03'	PRINT	X'09'	HLPTITLE
X'04'	IGCCNV	X'0A'	ALTNAME
X'05'	HLPRCD	X'0B'	ERRSFL
X'06'	HLPPNLGRP	X'0C'	WDWBORDER

File-Level Keywords (QDFKFILK)

File-level keywords. The displacement to this structure from the beginning of the appropriate section (file, record, or field) is from variable WDFFCAOF in structure QDFFCCOA.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFKFIL	Number of keywords to follow.
0	0		CHAR(*)	WDFKFILE	File-level keyword with parameters (see structure QDFKFLPM, "File-Level Keyword with Parameters (QDFKFLPM)").

File-Level Keyword with Parameters (QDFKFLPM)

File-level keyword with parameters. The number of entries in this structure is defined by variable WDFKFIL in structure QDFKFILK. This structure is defined at variable WDFKFILE in structure QDFKFILK. The structure is ARRAY(*).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKFLID	Keyword ID.
1	1		BIN(15)	WDFKFLST	Index into selection table (see structure QDFFSELT, "Selection Table (QDFFSELT)" on page 158). 1 indicates not optioned.
3	3		BIN(15)	WDFKFLRS	Response indicator minus one for files with INDARA keyword. Response indicator input buffer displacement for those without INDARA keyword. In either case, -1 represents no response indicator specified.
5	5		BIN(15)	WDFKFLP	Number of parameters to follow.
7	7		CHAR(*)	WDFKFLEX	Category 1 keyword parameter entries.

Category 1 Parameter Entry (QDFKFLPP)

Parameter entries for category 1. The number of entries in this structure is defined by variable WDFKFLP in structure QDFKFLPM. This structure is defined at variable WDFKFLEX in structure QDFKFLPM. The structure is ARRAY(*).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFKFLLN	Length of the following keyword parameter entry.
2	2		CHAR(*)	WDFKFLNM	Keyword parameter structure.

MSGLOC Keyword Structure (QDFKFLSZ)

MSGLOC keyword structure. Use this structure for the category 1 keyword that has a keyword ID of X'02' in structure QDFKFLPM ("File-Level Keyword with Parameters (QDFKFLPM)"). The number of entries in this structure is defined by variable WDFFSCR in structure QDFFBASE. This structure is defined at variable WDFKFLNM in structure QDFKFLPP. The structure is ARRAY(*).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKFLML	MSGLOC values.
2	2		CHAR(4)	*	Reserved.

IGCCNV Keyword Structure (QDFKICVP)

IGCCNV keyword structure. Use this structure for the category 1 keyword that has a keyword ID of X′04′ in structure QDFKFLPM ("File-Level Keyword with Parameters (QDFKFLPM)" on page 160). This structure is defined at variable WDFKFLNM in structure QDFKFLPP.

Note: The IGCCNV keyword generates additional internal records.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKICVN	IGCCNV format line number.
2	2		BIN(15)	WDFKICVK	IGCCNV format CF key.
4	4		BIN(15)	WDFKICVT	Index to internally generated record in the record format table.

HLPRCD Keyword Structure (QDFKHARD)

HLPRCD keyword structure. Use this structure for the category 1 keyword that has a keyword ID of X′05′ in structure QDFKFLPM ("File-Level Keyword with Parameters (QDFKFLPM)" on page 160). This structure is defined at variable WDFKFLNM in structure QDFKFLPP.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(10)	WDFKHRFM	Record format name.
10	A		CHAR(10)	WDFKHFIL	File name.
20	14		CHAR(10)	WDFKHLIB	File library name.

HLPPNLGRP Keyword Structure (QDFKHXPS)

HLPPNLGRP keyword structure. Use this structure for the category 1 keyword that has a keyword ID of X′06′ in structure QDFKFLPM ("File-Level Keyword with Parameters (QDFKFLPM)" on page 160). This structure is defined at variable WDFKFLNM in structure QDFKFLPP.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(10)	WDFKHXHG	Help panel group.
10	A		CHAR(10)	WDFKHXHL	Help panel group library name.
20	14		BIN(15)	WDFKHXML	Length of module name.
22	16		CHAR(*)	WDFKHXMN	Help module name.

HLPDOC Keyword Structure (QDFKHDOC)

HLPDOC keyword structure. Use this structure for the category 1 keyword that has a keyword ID of X′07′ in structure QDFKFLPM (“File-Level Keyword with Parameters (QDFKFLPM)” on page 160). This structure is defined at variable WDFKFLNM in structure QDFKFLPP.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		CHAR(10)	WDFKHDLA	Help text label name.
10	A		CHAR(12)	WDFKHDDO	Document name.
22	16		BIN(15)	WDFKHDFL	Length of folder name.
24	18		CHAR(*)	WDFKHDFD	Folder name.

HLPSCHIDX Keyword Structure (QDFKSIDX)

HLPSCHIDX keyword structure. Use this structure for the category 1 keyword that has a keyword ID of X′08′ in structure QDFKFLPM (“File-Level Keyword with Parameters (QDFKFLPM)” on page 160). This structure is defined at variable WDFKFLNM in structure QDFKFLPP.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		CHAR(10)	WDFKSIOB	Search index object name.
10	A		CHAR(10)	WDFKSILB	Search index object library name.

ALTNAME Keyword Structure (QDFKFALX)

ALTNAME keyword structure. Use this structure for the category 1 keyword that has a keyword ID of X′0A′ in structure QDFKFLPM. This structure is defined at variable WDFKFLNM in structure QDFKFLPP.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		BIN(15)	WDFKALT	Count of ALTNAME keywords.
2	2		CHAR(*)	WDFKAARY	Alternative names (see structure QDFKFALK, “ALTNAME Keyword Entry (QDFKFALK)”).

ALTNAME Keyword Entry (QDFKFALK)

ALTNAME keyword entry. This structure is defined at variable WDFKAARY in structure QDFKFALX.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		CHAR(10)	WDFKANME	ALTNAME keyword value (excluding apostrophes).
10	A		BIN(15)	WDFKAINX	Index to record format in record format table.

ERRSFL Keyword Structure (QDFKESFL)

ERRSFL keyword structure. Use this structure for the category 1 keyword that has a keyword ID of X'0B' in structure QDFKFLPM ("File-Level Keyword with Parameters (QDFKFLPM)" on page 160). This structure is defined at variable WDFKFLNM in structure QDFKFLPP.

Note: The ERRSFL keyword generates additional internal records.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		CHAR(10)	WDFKESCR	Subfile control record name.

WDWBORDER Keyword Structure (QDFKBODR)

WDWBORDER keyword structure. Use this structure for the category 1 keyword that has a keyword ID of X'0C' in structure QDFKFLPM ("File-Level Keyword with Parameters (QDFKFLPM)" on page 160). This structure is defined at variable WDFKFLNM in structure QDFKFLPP.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		CHAR(1)	WDFKCLOR	Values for *COLOR. X'00' indicates not specified, X'3A' indicates BLU, X'20' indicates GRN, X'22' indicates WHT, X'28' indicates RED, X'30' indicates TRQ, X'32' indicates YLW, X'38' indicates PNK.
1	1		CHAR(1)	WDFKDATR	Values for *DSPATR. Combination of two or more of these values: X'00' indicates no attribute X'30' indicates (*DSPATR CS), X'28' indicates (*DSPATR BL), X'24' indicates (*DSPATR UL), X'22' indicates (*DSPATR HI), X'21' indicates (*DSPATR RI), and X'27' indicates (*DSPATR ND).
2	2		CHAR(8)	WDFKCHRS	WDWBORDER characters in the following order: top-left corner, top horizontal, top-right corner, left vertical, right vertical, bottom-left corner, bottom horizontal, bottom-right corner. If not specified, eight entries of X'00' will occur.

Category 2 (Record-Level Command Key Keywords)

The following table shows the keyword ID that corresponds to the record-level command-key keywords. Use structure QDFKCKKE for category 2 keyword IDs X'01' through X'25' and X'30'.

ID	Keyword	ID	Keyword	ID	Keyword
X'01'	CA/CF01	X'0E'	CA/CF14	X'1A'	ROLLDOWN
X'02'	CA/CF02	X'0F'	CA/CF15	X'1B'	PRINT
X'03'	CA/CF03	X'10'	CA/CF16	X'1C'	HOME
X'04'	CA/CF04	X'11'	CA/CF17	X'1D'	CLEAR
X'05'	CA/CF05	X'12'	CA/CF18	X'1E'	HELP
X'06'	CA/CF06	X'13'	CA/CF19	X'20'	HLPRTN
X'07'	CA/CF07	X'14'	CA/CF20	X'21'	VLDCMDKEY
X'08'	CA/CF08	X'15'	CA/CF21	X'22'	ALTHELP
X'09'	CA/CF09	X'16'	CA/CF22	X'23'	ALTPAGEUP
X'0A'	CA/CF10	X'17'	CA/CF23	X'24'	ALTPAGEDWN
X'0B'	CA/CF11	X'18'	CA/CF24	X'25'	MNUBARSW
X'0C'	CA/CF12	X'19'	ROLLUP	X'30'	MNUCNL
X'0D'	CA/CF13				

Command Key Keyword Structure (QDFKCKKW)

Structure for command key keywords. The displacement to this structure from the beginning of the appropriate section (file, record, or field) is at variable WDFFCAOF in structure QDFFCCOA.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFKCKS	Number of entries in the array.
2	2		CHAR(*)	WDFKCKCM	Command key keyword entries (see structure QDFKCKKE, "Command Key Keyword Entries (QDFKCKKE)").

Command Key Keyword Entries (QDFKCKKE)

Command key keyword array. The number of entries in this structure is defined by variable WDFKCKS in structure QDFKCKKW. This structure is defined at variable WDFKCKCM in structure QDFKCKKW. The structure is ARRAY(*).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKCKID	Keyword ID.
1	1		BIN(15)	WDFKCKIN	Index into selection table. 1 indicates not optioned.
3	3		BIN(15)	WDFKCKRS	Response indicator minus one for files with INDARA keyword. Response indicator input buffer displacement for those without INDARA. -1 indicates no response indicator is specified.
5	5		CHAR(4)	WDFKCKMA	Key mask (ignored for VLDCMDKEY and HLPRTN keywords). The OR values for the key mask follow: X'80000000' CA/CF01, X'40000000' CA/CF02, X'20000000' CA/CF03, X'10000000' CA/CF04, X'08000000' CA/CF05, X'04000000' CA/CF06, X'02000000' CA/CF07, X'01000000' CA/CF08, X'00800000' CA/CF09, X'00400000' CA/CF10, X'00200000' CA/CF11, X'00100000' CA/CF12, X'00080000' CA/CF13, X'00040000' CA/CF14, X'00020000' CA/CF15, X'00010000' CA/CF16, X'00008000' CA/CF17, X'00004000' CA/CF18, X'00002000' CA/CF19, X'00001000' CA/CF20, X'00000800' CA/CF21, X'00000400' CA/CF22, X'00000200' CA/CF23, X'00000100' CA/CF24, X'00000080' ROLLUP, X'00000040' ROLLDOWN, X'00000020' PRINT, X'00000010' HOME, X'00000008' CLEAR, X'00000004' HELP.

Note: The following keywords use the first three bytes of the mask field (WDFKCKMA) for the command key mask for the command key associated with the keyword. If the keyword is specified without a command key, a default command key is used. In addition, these keywords use the last byte of WDFKCKMA for a special purpose. The last byte contains the AID byte which is returned when the command key associated with the keyword is pressed. For example, if ALTPAGEDWN is specified as ALTPAGEDWN(CF04), then the last byte of the mask is X'34'. If you need to OR the masks of these keywords with the masks of the other keywords, zero out the last byte of the mask first. The keywords and the AID bytes for the default command keys are:

Keyword	**Default AID byte**
ALTHELP	X'31'

Keyword	Default AID byte
ALTPAGEUP	X'37'
ALTPAGEDWN	X'38'
MNUBARSW	X'3A'
MNUCNL	X'3C'

Category 3 (OVERLAY-Related Keywords and PUTRETAIN)

The following table shows the keyword ID that corresponds to the OVERLAY-related keywords and PUTRETAIN. Not all keywords require a structure. There are no structures for keyword IDs X'02', X'03', X'04', X'05', X'06', X'08', and X'09'.

ID	Keyword	ID	Keyword
X'01'	OVERLAY	X'07'	PUTRETAIN
X'02'	PUTOVR	X'08'	PROTECT
X'03'	ERASEINP(*MDTON)	X'09'	INZINP
X'04'	MDTOFF(*UNPR)	X'10'	ERASE
X'05'	ERASEINP(*ALL)	X'11'	CLRL
X'06'	MDTOFF(*ALL)		

OVERLAY Keyword Structure (QDFKOVRR)

OVERLAY-related keywords. This structure is used if the keyword ID in structure QDFKFLPM ("File-Level Keyword with Parameters (QDFKFLPM)" on page 160) is X'01'. The displacement to this structure from the beginning of the appropriate section (file, record, and field) is at variable WDFFCAOF in structure QDFFCCOA.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKOLS	Number of keyword entries to follow.
2	2		CHAR(*)	WDFKOVRT	Array of keyword entries. Entries are contained in structure QDFKOVRE ("Keyword Structure (QDFKOVRE)") or QDFKOVRP ("OVERLAY and PUTRETAIN-Related Keyword Structure (QDFKOVRP)" on page 166).

Keyword Structure (QDFKOVRE)

Array structure for keywords. Use this structure for category 3 keywords that have a keyword ID of X'02', X'03', X'04', X'05', X'06', X'08', or X'09'. This structure is defined at variable WDFKOVRT in structure QDFKOVRR. The structure is ARRAY(*).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(1)	WDFKOLAD	Keyword ID.
1	1		BIN(15)	WDFKOLAN	Index into selection table. 1 indicates not optioned.

OVERLAY and PUTRETAIN-Related Keyword Structure (QDFKOVRP)

Structure for OVERLAY and PUTRETAIN-related keywords. Use this structure for category 3 keywords that have a keyword ID of X'01', X'07', X'10', or X'11'. This structure is defined at variable WDFKOVRT in structure QDFKOVRR. The structure is ARRAY(*).

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		CHAR(1)	WDFKOLID	Keyword ID.
1	1		BIN(15)	WDFKOLIN	Index into selection table. 1 indicates not optioned.
3	3		CHAR(*)	WDFKOLEX	Extra remaining portion of this category for ERASE and CLRL.

ERASE Keyword Structure (QDFKOLER)

ERASE keyword structure. Use this structure for the category 3 keyword that has a keyword ID of X'10'. This structure is defined at variable WDFKOLEX in structure QDFKOVRP.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		BIN(15)	WDFKOLE	Number of bytes to follow.
2	2		ARRAY(*) OF BIN(15)	WDFKOLAR	Indexes to the record format table for the format to be erased. 0, if format does not exist.

CLRL Keyword Structure (QDFKOLCL)

CLRL keyword structure. Use this structure for the category 3 keyword that has a keyword ID of X'11'. This structure is defined at variable WDFKOLEX in structure QDFKOVRP.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		BIN(15)	WDFKOLC	Number of bytes to follow (always 2).
2	2		BIN(15)	WDFKOLCN	Clear line number.

Category 4 (Record-Level Miscellaneous Keywords)

The following table shows the keyword ID that corresponds to the record-level miscellaneous keywords. Not all keywords require a structure. There are no structures for keyword IDs X'01', X'02', X'03', X'04', X'05', X'06', X'07', X'08', and X'09'.

Note: Keywords INDARA2 and RTNCSRLOC2 are internally generated.

ID	Keyword	ID	Keyword
X'01'	LOCK	X'09'	RMVWDW
X'02'	ALARM	X'0F'	DSPMOD
X'03'	BLINK	X'10'	CSRLOC
X'04'	LOGOUT	X'11'	INDARA
X'05'	ALWGPH	X'13'	SETOFF
X'06'	MSGALARM	X'15'	RTNCSRLOC
X'07'	RETLCKSTS	X'16'	MNUBARDSP

ID	Keyword	ID	Keyword
X’08’	USRRSTDSP		

Miscellaneous Record-Level Keywords (QDFKMSRL)

Miscellaneous record-level keywords. The displacement to this structure from the beginning of the appropriate section (file, record, or field) is at variable WDFFCAOF in structure QDFFCCOA.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFKMSS	Number of keywords to follow.
2	2		CHAR(*)	WDFKMSKW	Array of keyword entries. Entries are contained in structure QDFKMSAP (“Parameter Structure (QDFKMSAP)”).

Parameter Structure (QDFKMSAP)

Array structure for keywords with simple parameters. This structure is defined at variable WDFKMSKW in structure QDFKMSRL. The structure is ARRAY(*).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
2	2		CHAR(1)	WDFKMSAD	Keyword ID.
3	3		BIN(15)	WDFKMSAN	Index into selection table. 1 indicates not optioned.

Response Indicator Keyword Array (QDFKMSCP)

Miscellaneous record-level keywords. Use this structure for category 4 keywords that have a keyword ID of X’10’, X’11’, X’0F’, or X’13’. The displacement to this structure from the beginning of structure QDFFRINF is at variable WDFFINDO in structure QDFFRINF.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKMSID	Keyword ID.
1	1		BIN(15)	WDFKMSIN	Index into selection table. 1 indicates keyword is not optioned. For SETOFF, this contains the response indicator displacement.
3	3		CHAR(*)	WDFKMSEX	Additional structures for CSRLOC, INDARA, DSPMOD, and MNUBARDSP keywords. (See structures QDFKMSK1 on “CSRLOC Keyword Structure (QDFKMSK1),” QDFKMSK2 on “INDARA Keyword Structure (QDFKMSK2)” on page 168, QDFKMSK3 on “DSPMOD Keyword Structure (QDFKMSK3)” on page 168, and QDFKMSMBDSP on “MNUBARDSP Keyword Structure (QDFKMSMBDSP)” on page 169.)

CSRLOC Keyword Structure (QDFKMSK1)

Remaining portion of CSRLOC keyword. Use this structure for a category 4 keyword that has a keyword ID of X’10’. This structure is defined at variable WDFKMSEX in structure QDFKMSCP.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKMSLC	Length of data follows.
2	2		CHAR(1)	*	Reserved.
3	3		BIN(15)	WDFKMSRW	Output buffer displacement for row value field.
5	5		BIN(15)	WDFKMSCL	Output buffer displacement for column value field.
7	7		BIN(31)	WDFKMSFA	Index into name table for line number field.
11	B		BIN(31)	WDFKMSFB	Index into name table for position number field.

INDARA Keyword Structure (QDFKMSK2)

Remaining portion of INDARA keyword. Use this structure for a category 4 keyword that has a keyword ID of X'11'. This structure is defined at variable WDFKMSEX in structure QDFKMSCP.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKMSLN	Length of data that follows. Length may be zero.
2	2		ARRAY(*) OF BIN(15)	WDFKMSRI	List of response indicators that are referred to in this record (including SETOFF). The value is the indicator displacement (that is, the indicator number minus one). If an indicator is only an option indicator (and not a response indicator), this value is -1.

DSPMOD Keyword Structure (QDFKMSK3)

Remaining portion of DSPMOD keyword. Use this structure for a category 4 keyword that has a keyword ID of X'0F'. This structure is defined at variable WDFKMSEX in structure QDFKMSCP.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKMSDM	Index into display size array (valid values are 1 through 4).

RTNCSRLOC and RTNCSRLOC2 Keyword Structure (QDFKMSCLN)

Remaining portion of RTNCSRLOC and RTNCSRLOC2 keywords. Use this structure for a category 4 keyword that has a keyword ID of X'15'. This structure is defined at variable WDFKMSEX in structure QDFKMSCP.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKMSCRO	Displacement into input buffer to the hidden field that contains the name of the record that the cursor is on. Valid for RTNCSRLOC keyword or for the *RECNAME parameter of the RTNCSRLOC2 keyword.
0	0		BIN(15)	WDFKRCLR	Displacement into input buffer to the hidden field that contains the row number the cursor is on. Valid for *WINDOW or *MOUSE parameter of RTNCSRLOC2 keyword.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
2	2		BIN(15)	WDFKMSCFO	Displacement into input buffer to the hidden field that contains the name of the field that the cursor is on. Valid for RTNCSRLOC keyword or for the *RECNAME parameter of the RTNCSRLOC2 keyword.
2	2		BIN(15)	WDFKRCLC	Displacement into input buffer to the hidden field that contains the column number the cursor is on. Valid for *WINDOW or *MOUSE parameter of RTNCSRLOC2 keyword.
4	4		BIN(15)	WDFKMSCLO	Displacement into input buffer to the hidden field that contains the relative position into the field that the cursor is on. Valid for RTNCSRLOC keyword or for the *RECNAME parameter of the RTNCSRLOC2 keyword.
4	4		BIN(15)	WDFKCLWR	Displacement into input buffer to the hidden field that contains the row of the cursor relative to the active window or to the location of the cursor after the mouse button action has been processed. Valid for *WINDOW or *MOUSE parameter of RTNCSRLOC2 keyword.
6	6		BIN(15)	WDFKRCLWC	Displacement into input buffer to the hidden field that contains the column of the cursor relative to the active window or to the location of the cursor after the mouse button action has been processed. Valid for *WINDOW or *MOUSE parameter of RTNCSRLOC2 keyword and does not exist for the RTNCSRLOC keyword.
8	8		CHAR(1)	WDFKRCTYPE	The type of RTNCSRLOC format specified. X'00' indicates *RECNAME is specified, X'01' indicates *WINDOW is specified, and X'02' indicates *MOUSE is specified. This section is only valid for the RTNCSRLOC2 keyword and does not exist for the RTNCSRLOC keyword.
9	9		CHAR(1)	WDFKRCFLGS	Miscellaneous flags for the RTNCSRLOC2 keyword. This section is only valid for the RTNCSRLOC2 keyword and does not exist for the RTNCSRLOC keyword.
9	9	0	BIT(1)	WDFKRCLFMT2	1 indicates this is returning row-column information. 0 indicates this is returning record name and field name information. This section is only valid for the RTNCSRLOC2 keyword and does not exist for the RTNCSRLOC keyword.
9	9	1	BIT(7)	*	Reserved.

MNUBARDSP Keyword Structure (QDFKMSMBDSP)

Remaining portion of MNUBARDSP. Use this structure for a category 4 keyword that has a keyword ID of X'16'. This structure is defined at variable WDFKMSEX in structure QDFKMSCP.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(10)	WDFKMBDRCN	The name of the menu bar record that the MNUBARDSP keyword wants to display. If MNUBARDSP is on a MNUBAR keyword record, this field contains hexadecimal zeros.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
10	A		BIN(15)	WDFKMBDCFO	Displacement into input buffer to the hidden field that contains the number of the CHOICE specified by the user. -1, if not used.
12	C		BIN(15)	WDFKMBDPIO	If the PULLDOWN contains only the SNGCHCFLD keyword, this is the displacement into the input buffer to the hidden field that contains the input from the PULLDOWN. -1, if not used.

Category 6 Keywords (Record-Level Keywords)

The following table shows the keyword ID that corresponds to category 6 keywords. Both of these keyword IDs require a structure.

ID	Keyword
X’01’	INVITE
X’09’	FRCDTA

Record-level Keywords with selection array index (QDFRCAT06).

Record-level keywords with only an index into the selection table. The displacement to this structure from the beginning of the appropriate section (file, record, or field) is at variable WDFFCAOF in structure QDFFCCOA.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(1)	WDFKC6ID	Keyword ID.
1	1		BIN(15)	WDFKCINX	Index into selection table. 1 indicates not optioned.

Category 0B Keywords (File-Level Keywords with Parameters)

The following table shows the keyword ID that corresponds to category 0B keywords. Both of these keyword IDs require a structure.

ID	Keyword
X’01’	GRDATR
X’02’	HLPSHELF

File-Level Keywords with Parameters Structure (QDFK0BPR)

File-level keywords with parameters. The displacement to this structure from the beginning of the appropriate section (file, record, or field) is at variable WDFFCAOF in structure QDFFCCOA.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFK0BPCT	Number of keywords to follow.
2	2		CHAR(*)	WDFK0BKW	File-level keyword entries. See structure QDFK0BXWP (“File-Level Keyword Structure (QDFK0BXWP)” on page 171).

File-Level Keyword Structure (QDFK0BXWP)

File-level keyword structure. This structure is defined at variable WDFK0BKW in structure QDFK0BPR.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFK0BPID	Keyword ID.
1	1		BIN(15)	WDFK0BPIN	Index into selection table. 1 indicates keyword is not optioned.
3	3		BIN(15)	WDFK0BPLN	Length of parameter for keyword.
5	5		CHAR(*)	WDFK0BPEX	Extra remaining portion for keywords. The actual length is in variable WDFKMPLN in structure QDFKMRWP.

GRDATR Parameter Structure (QDFK0BGATR)

GRDATR parameter structure. Use this structure for category 0B keywords that have a keyword ID of X′01′. The structure is defined at variable WDFK0BPEX in structure QDFK0BXWP.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFK0BCLR	Color parameter. X′FE′ indicates not specified.
1	1		CHAR(1)	*	Reserved.
2	2		CHAR(1)	WDFK0BLT	Line type. X′FE′ indicates not specified.
3	3		CHAR(1)	*	Reserved.

HLPSHELF Parameter Structure (QDFKHBKPRM)

Structure for HLPSHELF parameters. Use this structure for category 0B keywords that have a keyword ID of X′02′. The structure is defined at variable WDFK0BPEX in structure QDFK0BXWP.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(8)	WDFKHBKNAM	Bookshelf name.

Category 17 (Record-Level Miscellaneous Keywords with Parameters)

The following table shows the keyword ID that corresponds to the record-level miscellaneous keywords with parameters. Not all keywords require a structure. There are no structures for keyword IDs X′03′, X′04′, and X′05′.

Note: HLP is an internal keyword generated when an H-specification is specified.

ID	Keyword	ID	Keyword
X′01′	HLP	X′05′	TIMER
X′02′	HLPSEQ	X′06′	PRINT
X′03′	HLPTITLE	X′07′	WDWBORDER
X′04′	HLPCLR	X′08′	WINDOW

Miscellaneous Record-Level Structure (QDFKMRPR)

Miscellaneous record-level with parameters. The displacement to this structure from the beginning of the appropriate section (file, record, or field) is at variable WDFFCAOF in structure QDFFCCOA.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFKRPS	Number of keywords to follow
2	2		CHAR(*)	WDFKRPKW	Record-level keyword entries (see structure QDFKMRWP, "Miscellaneous Record-Level Keywords (QDFKMRWP)").

Miscellaneous Record-Level Keywords (QDFKMRWP)

Miscellaneous record-level keywords. This structure is defined at variable WDFKRPKW in structure QDFKMRPR. The structure is ARRAY(*).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKMPID	Keyword ID.
1	1		BIN(15)	WDFKMPIN	Index into selection table. 1 indicates keyword is not optioned.
3	3		BIN(15)	WDFKMPLN	Length of parameter for keyword.
5	5		CHAR(*)	WDFKMPEX	Extra remaining portion for keywords. Actual length of parameter is specified in variable WDFKMPLN in this structure.

HLP Keyword Structure (QDFKHSTR)

Remaining portion of H-specification. Use this structure for a category 17 keyword that has a keyword ID of X'01'. This structure is defined at variable WDFKMPEX in structure QDFKMRWP.

Note: HLP is an internal keyword generated when an H-specification is specified.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFKH	Count of H-specifications.
2	2		CHAR(*)	WDFKHV	Variable part of parameter.

HLP Keyword Entry Structure (QDFKHPRM)

Entry for the internal HLP keyword. The total number of entries is contained in variable WDFKH in structure QDFKHSTR. This structure is defined at variable WDFKHV in structure QDFKHSTR. Displacements to subsequent entries are calculated using variable WDFKHOFS in structure QDFKHSTR. The structure is ARRAY(*).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFKHOFS	Length of this H-specification.
2	2		BIN(15)	WDFKHFLG	Flags for H-specification.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
2	2	0	BIT(1)	WDFKHBDY	If on, HLPBDY keyword is specified.
2	2	1	BIT(1)	WDFKHPRD	If on, HLPRCD keyword is specified. Structure QDFKHNMS (“HLPRCD Keyword Structure (QDFKHNMS)”) is defined at variable WDFKHVAR of this structure.
2	2	2	BIT(1)	WDFKHPNL	If on, HLPPNLGRP keyword is specified. Structure QDFKHPS (“HLPPNLGRP Keyword Structure (QDFKHPS)” on page 174) is defined at variable WDFKHVAR.
2	2	3	BIT(1)	WDFKHPDC	If on, HLPDOC keyword is specified. Structure QDFKHRDC (“HLPDOC Keyword Structure (QDFKHRDC)” on page 174) is defined at variable WDFKHVAR.
2	2	4	BIT(1)	WDFKDFHR	If on, file name on HLPRCD keyword is the default.
2	2	5	BIT(1)	WDFKHEXC	If on, HLPEXCLD keyword is specified.
2	2	6	BIT(1)	WDFKENPT	If on, CHOICE, MNUBAR, or PULLDOWN help was specified in this H-specification. This indicates enhanced display structure QDFKHARX (“HLPARA Keyword Enhanced Display Structure (QDFKHARX)” on page 175) is mapped at variable WDFKHEXT in structure QDFKHARA.
2	2	7	BIT(9)	*	Reserved.
4	4		BIN(15)	WDFKHSRO	Displacement to structure containing help source information. (See structure QDFKHNMS on “HLPRCD Keyword Structure (QDFKHNMS),” QDFKHPS on “HLPPNLGRP Keyword Structure (QDFKHPS)” on page 174, or QDFKHRDC on “HLPDOC Keyword Structure (QDFKHRDC)” on page 174.)
6	6		BIN(15)	WDFKHCRD	Selection string for help source on HLPRCD, HLPDOC, and HLPPNLGRP keywords. 1 indicates not specified or no indicator on keyword.
8	8		BIN(15)	WDFKHCBY	Selection string for HLPBDY keyword. 1 indicates keyword not specified or no indicator on keyword.
10	A		BIN(15)	WDFKHARO	Displacement to HLPARA information (see structure QDFKHARA, “HLPARA Keyword Structure (QDFKHARA)” on page 174).
12	C		BIN(15)	WDFKHCEX	Selection string for HLPEXCLD. 1 indicates keyword not specified or no indicator on keyword.
14	E		CHAR(3)	*	Reserved.
17	11		CHAR(*)	WDFKHVAR	Variable length parameters.

HLPRCD Keyword Structure (QDFKHNMS)

HLPRCD keyword structure. This structure is present only if variable WDFKHPRD in structure QDFKHPRM is set on. This structure is defined at variable WDFKHVAR in structure QDFKHPRM.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(10)	WDFKHRCD	Record format name.
10	A		CHAR(10)	WDFKHFLE	File name.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
20	14		CHAR(10)	WDFKHRLB	File library name. If file and library were not specified, these are the display file and library names. If file but not library is specified, the library name is *LIBL.

HLPPNLGRP Keyword Structure (QDFKHPS)

HLPPNLGRP keyword structure. This structure is present only if variable WDFKHPNL in structure QDFKHPRM is set on. This structure is defined at variable WDFKHVAR in structure QDFKHPRM.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(10)	WDFKHPNN	Help panel group name.
10	A		CHAR(10)	WDFKHPLB	Help panel group library name. If library was not specified, the library name is *LIBL.
20	14		BIN(15)	WDFKHMLN	Length of help module name.
22	16		CHAR(*)	WDFKHMN	Help module name.

HLPDOC Keyword Structure (QDFKHRDC)

HLPDOC keyword structure. This structure is present only if variable WDFKHPDC in structure QDFKHPRM is set on. This structure is defined at variable WDFKHVAR in structure QDFKHPRM.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(10)	WDFKHRLA	Help text label name.
10	A		CHAR(12)	WDFKHRDO	Document name.
22	16		BIN(15)	WDFKHRFL	Length of folder name.
24	18		CHAR(*)	WDFKHRFD	Folder name.

HLPARA Keyword Structure (QDFKHARA)

HLPARA keyword structure. This structure is repeated for each display size specified. The number of display sizes is defined by variable WDFFSCR in structure QDFFBASE. This structure is defined at variable WDFKHARO in structure QDFKHPRM. The structure is ARRAY(*).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(1)	WDFKHFRO	From row.
1	1		CHAR(1)	WDFKHFCO	From column.
2	2		CHAR(1)	WDFKHTRO	To row.
3	3		CHAR(1)	WDFKHTCO	To column.
4	4		CHAR(*)	WDFKHEXT	Enhanced display extension (see structure QDFKHARX, “HLPARA Keyword Enhanced Display Structure (QDFKHARX)” on page 175). This field is present only if variable WDFKENPT in structure QDFKHPRM is on.

HLPARA Keyword Enhanced Display Structure (QDFKHARX)

HLPARA enhanced display mapping. This structure is present only if variable WDFKHPRD in structure QDFKHPRM is on. This structure is defined at variable WDFKHEXT in structure QDFKHARA.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		CHAR(1)	WDFKHAFLG	Help area flags.
0	0	0	BIT(1)	WDFKHCHC	Choice-level help.
0	0	1	BIT(2)	*	Reserved.
0	0	3	BIT(1)	WDFKHFLDC	Choice-level help, with no choice number.
0	0	4	BIT(1)	WDFKHRC	If on, HLPARA(*RCD) keyword is specified.
0	0	5	BIT(3)	*	Reserved.
1	1		BIN(15)	WDFKHFLDI	Index to field on choice-level help.
3	3		CHAR(2)	WDFKHCHID	Choice number for choice-level help.

HLPSEQ Keyword Structure (QDFKHSEQ)

Remaining portion of HLPSEQ. Use this structure for a category 17 keyword that has a keyword ID of X’02’. This structure is defined at variable WDFKMPEX in structure QDFKMRWP.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		CHAR(10)	WDFKHSGN	Help group name.
10	A		BIN(15)	WDFKHSS	Help sequence number.
12	C		BIN(15)	WDFKHSIF	Index to first record in help group.
14	E		BIN(15)	WDFKHSIL	Index to last record in help group.
16	10		BIN(15)	WDFKHSIN	Index to next record in help group.
18	12		BIN(15)	WDFKHSIP	Index to previous record in help group.

PRINT Keyword Structure (QDFKPRTR)

PRINT keyword structure. Use this structure for a category 17 keyword that has a keyword ID of X’06’. This structure is defined at variable WDFKMPEX in structure QDFKMRWP.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		BIN(15)	WDFKRLRS	Response indicator; -1, if no response indicator.
2	2		BIN(15)	WDFKRLP	Number of parameters to follow.
4	4		CHAR(*)	WDFKPRTP	Remaining print structure.

Record-Level Print Parameters (QDFKPPRM)

Record-level print parameters. This structure is defined at variable WDFKMPEX in structure QDFKMRWP.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFKPLEN	Length of parameter to follow.
2	2		CHAR(*)	WDFKPFLN	Print file or library name.

WDWBORDER Keyword Structure (QDFKBRDR)

WDWBORDER keyword structure. Use this structure for a category 17 keyword that has a keyword ID of X'07'. This structure is defined at variable WDFKMPEX in structure QDFKMRWP.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKCOLR	Value for *COLOR. X'00' indicates not specified, X'3A' indicates BLU, X'20' indicates GRN, X'22' indicates WHT, X'28' indicates RED, X'30' indicates TRQ, X'32' indicates YLW, X'38' indicates PNK.
1	1		CHAR(1)	WDFKDSPA	Value for *DSPATR (combination of two or more of these values): X'00' indicates no attribute, X'30' indicates (*DSPATR CS), X'28' indicates (*DSPATR BL), X'24' indicates (*DSPATR UL), X'22' indicates (*DSPATR HI), X'21' indicates (*DSPATR RI), X'27' indicates (*DSPATR ND). If multiple values are specified, they are ORed together.
2	2		CHAR(8)	WDFKCHAR	Border characters in the following order: top-left corner, top horizontal, top-right corner, left vertical, right vertical, bottom-left corner, bottom horizontal, bottom-right corner.

Window Data Array Structure (QDFKWDTA)

Window data array. Use this structure for a category 17 keyword that has a keyword ID of X'08'. There is one array entry for each display size specified. This structure is defined at variable WDFKMPEX in structure QDFKMRWP. The structure is ARRAY(*).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(2)	WDFWDWB	Miscellaneous flags that describe the type of information provided by this window keyword.
0	0	0	BIT(1)	WDFKDEFN	1 indicates window definition; do not use variable WDFKWNAM in this structure. 0 indicates window reference; use WDFKWNAM.
0	0	1	BIT(1)	WDFKLINC	1 indicates actual line number provided. 0 indicates displacement to line number field provided. This field is not used if variable WDFKDEFN in this structure equals 0.
0	0	2	BIT(1)	WDFKLFL1	1 indicates line number field length is 1 digit long. 0 indicates line number is not 1 digit long. This field is not used if WDFKDEFN equals 0 or WDFKLINC equals 1.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0	3	BIT(1)	WDFKLFL2	1 indicates line number field length is 2 digits long. 0 indicates line number is not 1 digit long. This field is not used if WDFKDEFN equals 0 or WDFKLINC equals 1. **Note:** If WDFKLFL1 and WDFKLFL2 both equal 0, the field length is 3 digits.
0	0	4	BIT(1)	WDFKPOSC	1 indicates actual position number provided. 0 indicates displacement to position number field provided. This field is not used if WDFKDEFN equals 0.
0	0	5	BIT(1)	WDFKPFL1	1 indicates position number field length is 1 digit long. 0 indicates position number is not 1 digit long. This field is not used if WDFKDEFN equals 0 or WDFKPOSC equals 1.
0	0	6	BIT(1)	WDFKPFL2	1 indicates position number field length is 2 digits long. 0 indicates position number is not 1 digit long. This field is not used if WDFKDEFN equals 0 or WDFKPOSC equals 1. **Note:** If WDFKPFL1 and WDFKPFL2 both equal 0, the field length is 3 digits.
0	0	7	BIT(1)	WDFKDFTB	Default specified in place of first two parameters. Bits WDFKLINC and WDFKPOSC will also be set and WDFKLINW and WDFKPOSW will be set to 0.
1	1	0	BIT(1)	WDFKNMLN	If on, *NOMSGLIN parameter is specified. This window does not contain a message line.
1	1	1	BIT(1)	WDFKWRST	If on, *NORSTCSR parameter is specified. This window allows the function keys to work outside of the window.
1	1	2	BIT(6)	*	Reserved.
2	2		CHAR(10)	WDFKWNAM	Name of window definition record.
2	2		BIN(15)	WDFKLIN	Line number or displacement.
4	4		BIN(15)	WDFKPOS	Position number or displacement.
6	6		BIN(15)	WDFKLINW	Number of window lines in window.
8	8		BIN(15)	WDFKPOSW	Number of window positions in window.
10	A		CHAR(2)	*	Reserved.

Window Title Structure (QDFKWDWTTL)

Window title structure. This structure is defined at variable WDFKRPKW in structure QDFKMRPR.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(5)	WDFKWDWOVL	Miscellaneous record-level keywords. Structure QDFKMRWP (“Miscellaneous Record-Level Keywords (QDFKMRWP)” on page 172) overlays this field.
5	5		CHAR(1)	WDFKWTFLAGS	Miscellaneous flags that describe the type of information provided by this window title keyword.
5	5	0	BIT(1)	WDFKWTTXTF	1 indicates window title text is in a program-to-system field. 0 indicates window title text is a text literal.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
5	5	1	BIT(1)	WDFKWTCLRF	1 indicates window title color value is in a program-to-system field. 0 indicates window title color value is in the parameter.
5	5	2	BIT(1)	WDFKWTATRF	1 indicates window title attribute value is in a program-to-system field. 0 indicates window title attribute value is in the parameter.
5	5	3	BIT(1)	WDFKWTALNF	1 indicates window title alignment value is in a program-to-system field. 0 indicates window title alignment value is in the parameter.
5	5	4	BIT(1)	WDFKWTPOS	1 indicates window title goes in the bottom border. 0 indicates window title goes in the top border.
5	5	5	BIT(3)	*	Reserved.
6	6		BIN(15)	WDFKWTCLRPF	Buffer displacement to field that contains the color.
6	6		CHAR(1)	*	Reserved.
7	7		CHAR(1)	WDFKWTCOLOR	Value for *COLOR. X'00' indicates not specified, X'3A' indicates BLU, X'20' indicates GRN, X'22' indicates WHT, X'28' indicates RED, X'30' indicates TRQ, X'32' indicates YLW, and X'38' indicates PNK.
8	8		BIN(15)	WDFKWTDSPPF	Buffer displacement to the field that contains the attribute.
8	8		CHAR(1)	*	Reserved.
9	9		CHAR(1)	WDFKWTDSPA	Value for *DSPATR (combination of two or more of the values below). X'00' indicates no attribute, X'30' indicates (*DSPATR CS), X'28' indicates (*DSPATR BL), X'24' indicates (*DSPATR UL), X'22' indicates (*DSPATR HI), X'21' indicates (*DSPATR RI), and X'27' indicates (*DSPATR ND).
10	A		BIN(15)	WDFKWTALGN	Buffer displacement to the field that contains the alignment.
10	A		CHAR(1)	*	Reserved.
11	B		CHAR(1)	WDFKWTALIGN	Value for alignment. X'00' indicates not specified, X'01' indicates *left specified, X'02' indicates *CENTER specified, and X'03' indicates *RIGHT specified.
12	C		BIN(15)	WDFKWTTXTO	Buffer displacement to the field that contains the text if variable WDFKWTTXTF in this structure is on. Displacement is from structure QDFKWDWTTL ("Window Title Structure (QDFKWDWTTL)" on page 177).
14	E		BIN(15)	WDFKWTTXTL	Length of the text.
15	F		CHAR(10)	*	Reserved.
26	1A		CHAR(*)	WDFKWTTEXT	Text if entered on the parameter as a literal.

Mouse Button Structure (QDFKMB)

Mouse button structure. This structure is defined at variable WDFKMPEX in structure QDFKMRWP.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKMBFLAGS	Miscellaneous flags that describe the type of information provided by this mouse button keyword.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0	0	BIT(1)	WDFKMBTYPE	1 indicates two event mouse button definition. 0 indicates single event mouse button definition.
0	0	1	BIT(1)	WDFKMBTCSR	1 indicates move text cursor to mouse cursor. 0 indicates do not move text cursor to mouse cursor.
0	0	2	BIT(1)	WDFKMBQUE	1 indicates queue if keyboard locked. 0 indicates do not queue if keyboard locked.
0	0	3	BIT(1)	WDFKMBKRB	1 indicates marker box drawn. 0 indicates marker box not drawn.
0	0	4	BIT(4)	*	Reserved.
1	1		CHAR(1)	WDFKMBFIRST	Value for first event ID. X'01' indicates left button pressed, X'02' indicates left button released, X'03' indicates left button double-clicked, X'04' indicates right button pressed, X'05' indicates right button released, X'06' indicates right button double-clicked, X'07' indicates middle button pressed, X'08' indicates middle button released, X'09' indicates middle button double-clicked, X'0A' indicates shift left button pressed, X'0B' indicates shift left button released, X'0C' indicates shift left button double-clicked, X'0D' indicates shift right button pressed, X'0E' indicates shift right button released, X'0F' indicates shift right button double-clicked, X'10' indicates shift middle button pressed, X'11' indicates shift middle button released, and X'12' indicates shift middle button double-clicked.
2	2		CHAR(1)	WDFKMBSECOND	Value for second event ID. X'01' indicates left button pressed, X'02' indicates left button released, X'03' indicates left button double-clicked, X'04' indicates right button pressed, X'05' indicates right button released, X'06' indicates right button double-clicked, X'07' indicates middle button pressed, X'08' indicates middle button released, X'09' indicates middle button double-clicked, X'0A' indicates shift left button pressed, X'0B' indicates shift left button released, X'0C' indicates shift left button double-clicked, X'0D' indicates shift right button pressed, X'0E' indicates shift right button released, X'0F' indicates shift right button double-clicked, X'10' indicates shift middle button pressed, X'11' indicates shift middle button released, and X'12' indicates shift middle button double-clicked.
3	3		CHAR(1)	WDFKMBAID	AID code to be returned. X'31' through X'3C' indicates CA/CF01-12, X'70' through X'7F' indicates E00-E15, X'B1' through X'BC' indicates CA/CF13-24, X'BD' indicates CLEAR, X'F1' indicates ENTER, X'F3' indicates HELP, X'F4' indicates Roll Down, X'F5' indicates Roll Up, X'F6' indicates Print, and X'F8' indicates Home.

Category 18 (SFL Control Keywords)

The following table shows the keyword ID that corresponds to the following:

- The SFL control keywords that can be optioned
- The SFL control keywords processed by SFL and workstation

Not all keywords require a structure. There are no structures for keyword IDs X'01', X'02', X'03', X'04', X'05', X'06', X'07', X'08', and X'09'.

ID	Keyword	ID	Keyword
X'01'	SFLDSP	X'09'	SFLDROP
X'02'	SFLDSPCTL	X'0D'	SFLMSG
X'03'	SFLINZ	X'0E'	SFLMSGID
X'04'	SFLDLT	X'0F'	SFLEND(*MORE)
X'05'	SFLCLR	X'10'	SFLCSRRRN
X'06'	SFLEND	X'11'	SFLMODE
X'07'	SFLNXTCHG	X'12'	SFLEND(*SCRBAR)
X'08'	SFLFOLD		

SFL Keyword Structure (QDFKSCSF)

SFL keyword structure. The displacement to this structure from the beginning of the appropriate section (file, record, or field) is at variable WDFFCAOF in structure QDFFCCOA.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFKSCS	Number of keywords to follow.
2	2		CHAR(*)	WDFKSCSE	SFL keyword entries (see structure QDFKSCCP, "SFL Keyword Entry (QDFKSCCP)").

SFL Keyword Entry (QDFKSCCP)

SFL keyword entry. This structure is defined at variable WDFKSCSE in structure QDFKSCSF.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKSCID	Keyword ID.
1	1		BIN(15)	WDFKSCIN	Index into selection table. 1 indicates not optioned.
3	3		CHAR(*)	WDFKSCEX	Extra remaining portion of this category.

SFLMSG and SFLMSGID Keyword Structure (QDFKSCSM)

Structure for SFLMSG and SFLMSGID keywords. Use this structure for a category 18 keyword that has a keyword ID of X'0D' or X'0E'. This structure is defined at variable WDFKSCEX in structure QDFKSCCP.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFKSCRS	Response indicator minus one for files with INDARA keyword. Response indicator input buffer displacement for those without INDARA keyword. In either case, -1 represents no response indicator is specified. For SFLMSGID, this field contains hexadecimal zeros.
2	2		CHAR(1)	WDFKSCTY	Parameter type (for SFLMSG). X'00' indicates character, X'08' indicates DBCS.
3	3		BIN(15)	WDFKSCLN	Length of data to follow.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
5	5		CHAR(*)	WDFKSCTX	Parameters of text or data.

SFLMSGID Keyword Structure (QDFKSCSI)

SFLMSGID keyword structure. Use this structure for a category 18 keyword that has a keyword ID of X'0E'. The first three fields are specified. If the user does not specify a library, *LIBL is the default. This structure is defined at variable WDFKSCTX in structure QDFKSCSM.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(7)	WDFKSCMI	Message ID.
7	7		CHAR(10)	WDFKSCMF	Message file name.
17	11		CHAR(10)	WDFKSCML	Message library name.
27	1B		BIN(15)	WDFKSCDL	Message data field length.
29	1D		BIN(15)	WDFKSCMO	Output buffer displacement to message data field.

SFLEND(*MORE) Keyword Structure (QDFKSFLM)

SFLEND(*MORE) keyword structure. Use this structure for a category 18 keyword that has a keyword ID of X'0F'. This structure is defined at variable WDFKSCEX in structure QDFKSCCP.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(12)	WDFKMORE	Text to be used for `More...`.
12	C		CHAR(12)	WDFKBOTT	Text to be used for `Bottom`.

SFLEND(*SCRBAR) Keyword Structure (QDFKSFLS)

SFLEND(*SCRBAR) keyword structure. Use this structure for a category 18 keyword that has a keyword ID of X'12'. This structure is defined at variable WDFKSCEX in structure QDFKSCCP.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKSFLENDSPM	Second parameter value for SFLEND(*SCRBAR). X'00' indicates *SCRBAR, X'01' indicates *MORE, and X'02' indicates *PLUS.
1	1		CHAR(12)	WDFKSFLMORE	Text to be used for `More...`.
13	D		CHAR(12)	WDFKSFLBOTT	Text to be used for `Bottom`.

SFLCSRRRN Keyword Structure (QDFKCSRRRN)

SFLCSRRRN keyword structure. Use this structure for a category 18 keyword that has a keyword ID of X'10'. This structure is defined at VARIABLE WDFKSCEX in structure QDFKSCCP.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFKCSRNO	Displacement into input buffer to the hidden field containing the subfile relative record number of where the cursor is located.

SFLMODE Keyword Structure (QDFKMODE)

SFLMODE keyword structure. Use this structure for a category 18 keyword that has a keyword ID of X’11’. This structure is defined at variable WDFKSCEX in structure QDFKSCCP.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFKSCMDO	Displacement into input buffer to the hidden field containing the mode of the subfile. If set to 0, the MODE keyword was not specified.

Category 20 (Screen-Attribute-Related Keywords)

The following table shows the keyword ID that corresponds to the screen-attribute-related keywords. None of these keyword IDs require a structure.

ID	Keyword	ID	Keyword
X’01’	COLOR	X’07’	DSPATR(HI)
X’04’	DSPATR(CS)	X’08’	DSPATR(RI)
X’05’	DSPATR(BL)	X’09’	DSPATR(ND)
X’06’	DSPATR(UL)	X’0A’	DSPATR(PC)

Screen Attribute Keyword Structure (QDFKSASA)

Structure for screen attribute keywords. The displacement to this structure from the beginning of the appropriate section (file, record, or field) is at variable WDFFCAOF in structure QDFFCCOA.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFKSAS	Number of keyword entries to follow.
2	2		CHAR(*)	WDFKSASE	Screen attribute keywords (see structure QDFKSAPM, “Screen Attribute Keyword Array (QDFKSAPM)”).

Screen Attribute Keyword Array (QDFKSAPM)

Array of screen attribute keywords. This structure is defined at variable WDFKSASE in structure QDFKSASA. The structure is ARRAY(*).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKSAIA	Keyword ID.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
1	1		BIN(15)	WDFKSAII	Index into selection table. For color keywords, 1 indicates keyword is not optioned. Unoptioned DSPATR keywords are not in this category but are indicated in the screen attribute variable WDFFSA in structure QDFFFDPD.
3	3		CHAR(1)	WDFKSAOA	OR value color. X'20' indicates GRN, X'22' indicates WHT, X'28' indicates RED, X'30' indicates TRQ, X'32' indicates YLW, X'38' indicates PNK, and X'3A' indicates BLU. DSPATR. X'30' indicates CS, X'28' indicates BL, X'24' indicates UL, X'22' indicates HI, X'21' indicates RI, and X'27' indicates ND.

Category 21 Keywords

The following table shows the keyword ID that corresponds to category 21 keywords. All of these keyword IDs require a structure.

ID	Keyword
X'01'	DSPATR(PR)
X'02'	DUP
X'03'	DSPATR(MDT)
X'04'	AUTO(RA)
X'05'	CHECK(ME)

FFW and FCW Keyword Structure (QDFKFFWR)

Structure for field format word (FFW) and field control word (FCW) keywords. The displacement to this structure from the beginning of the section is an entry in the table at variable WDFFCAOF in structure QDFFCCOA.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKFWS	Number of keywords to follow.
2	2		CHAR(*)	WDFKFWEN	FFW keyword entries (see structure QDFKCHKP, "FFW Keyword Structure (QDFKCHKP)").

FFW Keyword Structure (QDFKCHKP)

Structure for FFW-related keywords. Use this structure for category 21 keywords that have keyword IDs of X'01', X'02', X'03', X'04', and X'05'. This structure is defined at variable WDFKFWEN in structure QDFKFFWR. The structure is ARRAY(*).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(1)	WDFKFWID	Keyword ID.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
1	1		BIN(15)	WDFKFWIN	Index into selection table. 1 indicates keyword is not optioned. Keywords DSPATR(PR ∣ MDT), DUP, AUTO(RA), and CHECK(ME), which are not optioned, are not in this category but are indicated in the FFW WDFFFWFW VARIABLE in structure QDFFFDIC.
3	3		CHAR(2)	WDFKFWOV	OR value for FFW. X'6000' indicates DSPATR(PR), X'5000' indicates DUP, X'4800' indicates DSPATR(MDT), X'4080' indicates AUTO(RA), X'4008' indicates CHECK(ME).

Category 22 (Miscellaneous Field-Level Keywords)

The following table shows the keyword ID that corresponds to miscellaneous field-level keywords. Not all keywords require a structure. There are no structures for keyword IDs X'01', X'02', X'03', X'04', X'05', X'06' and X'07'.

ID	Keyword	ID	Keyword
X'01'	PUTRETAIN	X'10'	MSGID
X'02'	OVRDTA	X'15'	ERRMSG
X'03'	OVRATR	X'16'	ERRMSGID
X'04'	BLANKS	X'17'	DSPATR(PFLD)
X'05'	CHANGE	X'18'	DATTIMFMT
X'06'	DUP	X'19'	DATTIMSEP
X'07'	DUP	X'1A'	DATE (special value)
		X'1B'	MAPVAL

Miscellaneous Field-Level Keyword Structure (QDFKMFDK)

Miscellaneous field-level keywords. The displacement to this structure from the beginning of the appropriate section (file, record, or field) is at variable WDFFCAOF in structure QDFFCCOA.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKMFS	Number of keywords to follow.
2	2		BIN(15)	WDFKMFNO	Index into field index table of next field that has either ERRMSG or ERRMSGID. 0, if this the last one in the record or none exist.
4	4		CHAR(*)	WDFKMFEN	Field-level keyword entry (see structure QDFKMFDP, "Field-Level Keyword Structure (QDFKMFDP)").

Field-Level Keyword Structure (QDFKMFDP)

Field-level keyword parameters. The number of keyword parameters is contained in variable WDFKMFS in structure QDFKMFDK. This structure is defined at variable WDFKMFEN in structure QDFKMFDK.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(1)	WDFKMFID	Keyword ID.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
1	1		BIN(15)	WDFKMFIN	Index into selection table. 1 indicates keyword is not optioned.
3	3		CHAR(*)	WDFKMFEX	Extra remaining portion of this category.

Response Indicator Structure (QDFKMFRS)

Response indicator. This structure is defined at variable WDFKMFEX in structure QDFKMFDP.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKMFRP	Response indicator minus one for files with INDARA keyword. Response indicator input buffer displacement for those without INDARA. In either case, -1 represents no response indicator specified. For ERRMSGID, this field contains hexadecimal zeros.
2	2		CHAR(*)	WDFKMFEE	ERRMSG and ERRMSGID data.

ERRMSG and ERRMSGID Keyword Structure (QDFKMFEM)

Structure for ERRMSG and ERRMSGID keywords. Use this structure for category 22 keywords that have keyword IDs of X'15' and X'16'. This structure is defined at variable WDFKMFEE in structure QDFKMFRS.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(1)	WDFKMFTP	Type of parameter (for ERRMSG). X'00' indicates character, X'08' indicates DBCS.
1	1		BIN(15)	WDFKMFEL	Length of data to follow.
3	3		CHAR(*)	WDFKMFTX	ERRMSG and ERRMSGID data.

ERRMSGID Keyword Structure (QDFKMFSI)

ERRMSGID keyword structure. Use this structure for category 22 keywords that have a keyword ID of X'16'. The first three fields are specified. If the user does not specify library, *LIBL is the default. This structure is defined at variable WDFKMFTX in structure QDFKMFEM.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(7)	WDFKMFMI	Message ID.
7	7		CHAR(10)	WDFKMFMF	Message file name.
17	11		CHAR(10)	WDFKMFML	Message library name.
27	1B		BIN(15)	WDFKMFDL	Message data field length.
29	1D		BIN(15)	WDFKMFMO	Output buffer displacement to message data field.

MSGID Keyword Common Structure (QDFKMFMV)

MSGID keyword structure. Use this structure for category 22 keywords that have a keyword ID of X’10’. This structure is defined at variable WDFKMFEX in structure QDFKMFDP.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKMFMM	Mode of MSGID keyword. X’01’ indicates prefix specified, X’02’ indicates no prefix specified, X’03’ indicates constant message ID, and X’04’ indicates none. For mode 1 or 2, use this structure. For mode 3, use structure QDFKMFM3 (“Type Three MSGID Keyword Structure (QDFKMFM3)”). For mode 4, use structure QDFKMFM4 (“Type Four MSGID Keyword Structure (QDFKMFM4)” on page 187).
1	1		CHAR(3)	WDFKMFMP	Message prefix. This field contains hexadecimal zeros when variable WDFKMFMM in this structure equals X’02’. When WDFKMFMM equals X’03’, this field is the same value as variable WDFKMF1 in structure QDFKMFM3.
4	4		BIN(15)	WDFKMFFL	Message file length. This is the length of the field that contains the message file name. This field is not set when the file is a constant or special value.
6	6		CHAR(10)	WDFKMFFV	Message file name set when a constant or special value is specified for the message file.
16	10		CHAR(10)	WDFKMFLV	Message file library name set when a constant or no library is specified for the message file.
26	1A		ARRAY(3) OF BIN(15)	WDFKMFB	Three output buffer displacements to the fields in the following order: (1) MSGID field, (2) message file field, and (3) message library field. For message file or message library, X’FFFF’ indicates constants. When variable WDFKMFMM equals X’03’, the MSGID field equals X’FFFF’.
32	20		ARRAY(3) OF BIN(31)	WDFKMFNT	Indexes to a field in structure QDFFNTB (“Field Name Table (QDFFNTB)” on page 152) in the same order as variable WDFKMFB in this structure. If not specified, set to hexadecimal zeros.
43	2B		CHAR(*)	WDFKMFMX	Extension for type three MSGID keyword structure (only when variable WDFKMFMM equals X’03’).

Type Three MSGID Keyword Structure (QDFKMFM3)

Structure for MSGID keyword when variable WDFKMFMM in structure QDFKMFMV equals X’03’. This structure is defined at variable WDFKMFMX in structure QDFKMFMV.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(3)	WDFKMF1	Constant message number. Message prefix (same as WDFKMFMP in structure QDFKMFMV).
3	3		CHAR(4)	WDFKMF2	Message ID.

Type Four MSGID Keyword Structure (QDFKMFM4)

Structure for MSGID(*NONE) keyword when variable WDFKMFMM in structure QDFKMFMV equals X'04'. This structure is defined at variable WDFKMFEX in structure QDFKMFDP.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		CHAR(1)	WDFKMFD4	Mode of MSGID keyword. MSGID(*NONE) equals X'04'.

DSPATR Keyword Structure (QDFKDFLD)

DSPATR keyword structure. Use this structure for a category 22 keyword that has a keyword ID of X'17'. This structure is defined at variable WDFKMFEX in structure QDFKMFDP.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		BIN(15)	WDFKDFLO	Displacement to attribute field.

DATTIMFMT Keyword Structure (QDFK_DATTIM_Format)

The DATFMT or TIMFMT keyword structure. Use this category 22 keyword structure for ID X'18'. This structure is defined at variable WDFKMFEX in structure QDFKMFDP.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		BIN(15)	WDFK_DATTIM _Fmt	Format used for a field with the date or time data type. X'01' indicates *JOB, X'02' indicates *MDY, X'03' indicates *DMY, X'04' indicates *YMD, X'05' indicates *JUL, X'06' indicates *ISO, X'07' indicates *USA, X'08' indicates *EUR, X'09' indicates *JIS, X'0A' indicates *HMS.

DATTIMSEP Keyword Structure (QDFK_DATTIM_Separator)

The DATSEP or TIMSEP keyword structure. Use this category 22 keyword structure for ID X'19'. This structure is defined at variable WDFKMFEX in structure QDFKMFDP.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		CHAR(1)	WDFK_DATTIM _Sep	Separator used for a field with the date or time data type. The separator can be a period (.), comma (,), slash (/), dash (-), colon (:), blank ( ) or (J) to indicate *JOB.

DATE Keyword Structure (QDFK_DATEP)

The DATE (with parameters) keyword structure. Use this category 22 keyword structure for ID X'1A'. This structure is defined at variable WDFKMFEX in structure QDFKMFDP.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0	0	BIT(1)	WDFK_DATE_SYS	If on, the *SYS parameter is specified on the DATE keyword.
0	0	1	BIT(1)	WDFK_DATE_YY	If on, the *YY parameter is specified on the DATE keyword.
0	0	2	BIT(1)	WDFK_DATE _EDTCDEY	If on, the EDTCDE(Y) keyword was specified with the DATE keyword.
0	0	2	BIT(5)	*	Reserved.

MAPVAL Keyword Structure (QDFK_MAPVAL)

The MAPVAL keyword structure. Use this category 22 keyword structure for ID X'1B'. This structure is defined at variable WDFKMFEX in structure QDFKMFDP.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFK_MAPVAL _Count	Number of MAPVAL keyword parameters.
2	2		BIN(15)	WDFK_MAPVAL _Length	Length of each MAPVAL keyword parameter.
4	4		CHAR(*)	WDFK_MAPVAL _Length	List of MAPVAL keyword parameters. Length of this structure is WDFK_MAPVAL_Count * WDFK_MAPVAL_Length.

Category 23 (DFT Keyword)

The following table shows the keyword ID that corresponds to the DFT keyword. Not all keywords require a structure. There are no structures for keyword IDs X'01' and X'03'.

ID	Keyword
X'01'	DFT
X'02'	MSGCON
X'03'	DFTVAL
X'04'	HTML

Category 23 Keyword Structure (QDFKDFT)

Category 23 keyword structure. The displacement to this structure from the beginning of the appropriate section (file, record, or field) is at variable WDFFCAOF in structure QDFFCCOA.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKDFS	Number of keywords to follow.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
2	2		CHAR(*)	WDFKDFPE	Category 23 parameter entries (see structure QDFKDFPM, "Category 23 Keyword Parameters (QDFKDFPM)").

Category 23 Keyword Parameters (QDFKDFPM)

Category 23 keyword parameters. This structure is defined at variable WDFKDFPE in structure QDFKDFT. Displacements to subsequent entries are calculated using variable WDFKDFLN in this structure. The structure is ARRAY(*).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKDFID	Keyword ID.
1	1		CHAR(1)	WDFKDFTY	Parameter type. X'00' indicates character, X'03' indicates graphic literal, and X'08' indicates DBCS.
2	2		BIN(15)	WDFKDFIN	Index into selection table. 1 indicates keyword is not optioned.
4	4		BIN(15)	WDFKDFLN	Length of data to follow. For the MSGCON keyword, this value is only the length of the message text.
6	6		CHAR(*)	WDFKDFDF	Parameter for MSGCON keyword.

MSGCON Keyword Structure (QDFKDFMM)

MSGCON keyword structure. Use this structure for a category 23 keyword that has a keyword ID of X'02'. All three fields are specified. This structure follows variable WDFKDFDF in structure QDFKDFPM.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(7)	WDFKDFMI	Message ID.
7	7		CHAR(10)	WDFKDFMF	Message file name.
17	11		CHAR(10)	WDFKDFML	Message file library name. If the user does not specify library, *LIBL is the default.

HTML Keyword Structure (QDFKDFHTML)

HTML keyword structure. Use this structure for a category 23 keyword that has a keyword ID of X'04'. This structure is defined at variable WDFKDFDF in structure QDFKDFPM.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKDFHFLAG	Miscellaneous HTML tags
0	0	0	BIT(1)	WDFKDFHPFLD	If the bit is set on, then a program-to-system field was specified on the HTML keyword.
0	0	1	BIT(7)	*	Reserved.
1	1		BIN(15)	WDFKDFHLEN	Length of HTML text string or program-to-system field length.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
3	3		BIN(15)	WDFKDFHOFF	Offset to the program-to-system field from the start of the output buffer. This field is set to zero if a program-to-system field is not used.
5	5		CHAR(*)	WDFKDFHTMLTEXT	HTML text string. The length of this field is given in WDFKDFHLEN.

Category 24 (Field-Level Editing and Time Keywords)

The following table shows the keyword ID that corresponds to the field-level editing and time keywords.

ID	Keyword
X'01'	EDTWRD
X'02'	EDTCDE

EDIT Keyword Structure (QDFKEDTR)

Structure for editing date and time type keywords. The displacement to this structure from the beginning of the appropriate section (file, record, and field) is at variable WDFFCAOF in structure QDFFCCOA.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFKEDS	Number of keywords to follow.
2	2		CHAR(*)	WDFKEDKW	EDIT keyword parameters (see structure QDFKEDTP, "EDIT Keyword Structure (QDFKEDTP)").

EDIT Keyword Structure (QDFKEDTP)

EDIT keyword structure. Use this structure for category 24 keywords that have keyword IDs of X'01' and X'02'. This structure is defined at variable WDFKEDKW in structure QDFKEDTR. The structure is ARRAY(*).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKEDID	Keyword ID.
1	1		CHAR(1)	WDFKEDSY	Zero suppress or fill character for the EDTCDE keyword.
2	2		BIN(15)	WDFKEDML	Length of the edit mask.
4	4		CHAR(1)	*	Reserved.
5	5		CHAR(*)	WDFKEDMS	The edit mask for the EDTCDE and EDTWRD keywords.

Category 25 (GET Validation Keywords)

The following table shows the keyword ID that corresponds to the GET validation keywords.

ID	Keyword	ID	Keyword	ID	Keyword
X'01'	RANGE	X'07'	CMP(LE)	X'0D'	CHECK(M11)

ID	Keyword	ID	Keyword	ID	Keyword
X'02'	VALUES	X'08'	CMP(LT)	X'0E'	CHECK(VN)
X'03'	CMP(GT)	X'09'	CMP(NL)	X'0F'	CHECK(VNE)
X'04'	CMP(GE)	X'0A'	CMP(NG)	X'10'	CHECK(M10F)
X'05'	CMP(EQ)	X'0B'	CHKMSGID	X'11'	CHECK(M11F)
X'06'	CMP(NE)	X'0C'	CHECK(M10)		

Validity Checking Keyword Structure (QDFKVAKW)

Structure for validity-checking type keywords. The displacement to this structure from the beginning of the appropriate section (file, record, or field) is at variable WDFFCAOF in structure QDFFCCOA.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFKVAL	Number of validity checking structures to follow. This value is 1. If the CHKMSGID keyword is present, variable WDFKCMID in structure QDFKVARL is set on and structure QDFKCKMI ("CHKMSGID Keyword Structure (QDFKCKMI)" on page 192) is present.
2	2		CHAR(*)	WDFKVACK	Validity checking keywords (see structure QDFKVARL, "Validity Checking Keywords (QDFKVARL)").

Validity Checking Keywords (QDFKVARL)

Validity-checking type keywords. Use this structure for category 25 keywords that have keyword IDs of X'01' through X'11'. This structure is defined at variable WDFKVACK in structure QDFKVAKW.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKVALC	Miscellaneous flags.
0	0	0	BIT(1)	WDFKM10F	If on, CHECK(M10F) keyword is specified.
0	0	1	BIT(1)	WDFKM11F	If on, CHECK(M11F) keyword is specified.
0	0	2	BIT(1)	WDFKCHVN	If on, CHECK(VN) keyword is specified.
0	0	3	BIT(1)	WDFKCHVE	If on, CHECK(VNE) keyword is specified. f
0	0	4	BIT(1)	WDFKCMID	If on, CHKMSGID keyword is specified. The structure QDFKCKMI ("CHKMSGID Keyword Structure (QDFKCKMI)" on page 192) is present.
0	0	5	BIT(1)	WDFKM10	If on, CHECK(M10) keyword is specified.
0	0	6	BIT(1)	WDFKM11	If on, CHECK(M11) keyword is specified.
0	0	7	BIT(1)	*	Reserved.
1	1		CHAR(1)	WDFKVALB	Flags for CMP, RANGE, and VALUE keywords.
1	1	0	BIT(4)	WDFKVAL	B'0000' indicates NONE, B'0001' indicates RANGE, B'0010' indicates VALUE, B'0011' indicates CMP(GT), B'0100' indicates CMP(GE or NL), B'0101' indicates CMP(EQ), B'0110' indicates CMP(NE), B'0111' indicates CMP(LE or NG), and B'1000' indicates CMP(LT).
1	1	4	BIT(4)	*	Reserved.
2	2		CHAR(1)	*	Reserved.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
3	3		CHAR(1)	WDFKVATP	Type of parameters. If a parameter has graphic literals, this value is X'03'. If any parameter has DBCS literals, this value is X'08'. Otherwise, this field contains 0.
4	4		BIN(15)	WDFKLAP	Number of parameters.
6	6		BIN(15)	WDFKLATP	Total length of parameters. (Each parameter length is wdffplen in structure QDFFFNAM.)
8	8		CHAR(*)	WDFKAPRM	Validity checking keywords (see structure QDFKCKMI, "CHKMSGID Keyword Structure (QDFKCKMI)"). This structure is present if variable WDFKCMID in this structure is on.

CHKMSGID Keyword Structure (QDFKCKMI)

CHKMSGID keyword structure. This structure is present if variable WDFKCMID in structure QDFKVARL is on. This structure is defined at variable WDFKAPRM in structure QDFKVARL.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(7)	WDFKCKMD	Message identifier.
7	7		CHAR(10)	WDFKCKMF	Message file name.
17	11		CHAR(10)	WDFKCKML	Message file library name.
27	1B		BIN(15)	WDFKCKDL	Message data length. Length of field that contains message data name. If no message data name exists, this is set to 0.
29	1D		BIN(15)	WDFKCKB	Output buffer displacement to message data field.

Category 26 (Field-Level Keywords for CUA Constructs)

The following table shows the keyword ID that corresponds to field-level keywords for Common User Access(R) (CUA(R)) constructs. All of these keyword IDs require a structure.

Note: CHCFLD is generated internally whenever the CHOICE, MNUBARCHC, or PSHBTNCHC keyword is specified.

ID	Keyword	ID	Keyword
X'01'	CHCFLD	X'06'	ENTFLDATR
X'02'	MNUBARSEP	X'07'	FLDCSRPRG
X'03'	CHCAVAIL	X'08'	CNTFLD
X'04'	CHCSLT	X'09'	EDTMSK
X'05'	CHCUNAVAIL		

Field-Level CUA Keyword Structure (QDFKFCPR)

Structure for field-level CUA keywords with parameters. The displacement to this structure from the beginning of the appropriate section (file, record, or field) is at variable WDFFCAOF in structure QDFFCCOA.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKFCS	Number of keywords to follow.
2	2		CHAR(*)	WDFKFCKW	Field-level CUA keywords (see structure QDFKFC, "Field-Level CUA Keywords (QDFKFC)").

Field-Level CUA Keywords (QDFKFC)

CUA keyword structure. This structure is defined at variable WDFKFCKW in structure QDFKFCPR. The structure is ARRAY(*).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(1)	WDFKFCID	Keyword ID.
1	1		BIN(15)	WDFKFCIN	Index into selection table. 1 indicates keyword is not optioned.
3	3		BIN(15)	WDFKFCLN	Length of parameter for keyword.
5	5		CHAR(*)	WDFKFCEX	Extra remaining portion for keywords.

CHCFLD Keyword Structure (QDFKCHC)

CHCFLD keyword structure. Use this structure for a category 26 keyword that has a keyword ID of X'01'. This structure is defined at variable WDFKFCEX in structure QDFKFC.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKCHCCT	Number of choice entries to follow.
2	2		CHAR(10)	WDFKCHCFN	Name of the choice field.
Note: The following two fields are maximum dimensions of the choice field. Set only for single- and multiple-choice selection fields and push-button fields.					
12	C		BIN(15)	WDFKCHCR	Maximum number of rows.
14	E		BIN(15)	WDFKCHCC	Maximum number of columns.
Note: The following two fields are selection characters to be used for multiple-choice selection fields.					
16	10		CHAR(1)	WDFKSELCHAR1	First character to be used.
17	11		CHAR(1)	WDFKSELCHAR2	Second character to be used.
18	12		CHAR(1)	*	Reserved.
19	13		CHAR(*)	WDFKCHCS	Choice entries.

CHCFLD Keyword Header Expansion Structure (QDFKCHCHDREXP)

CHCFLD header expansion structure. If bit WDFFCHCHDHEXP in structure QDFFXFDP is on, this structure is used. This structure is defined at variable WDFKCHCS in structure QDFKCHC.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKCHCEXPLEN	Length of the CHCFLD header expansion area.
2	2		BIN(15)	WDFKROWCOL	Value with either the *NUMROW or *NUMCOL parameter. Valid only if variable WDFKHORIZ in this structure is on.
4	4		BIN(15)	WDFKGUTTER	Value with the *GUTTER parameter.
6	6		CHAR(2)	WDFKFLAGS	Miscellaneous flags to describe how the choice fields were specified.
6	6	0	BIT(1)	WDFKRSTCSR	If on, *RSTCSR parameter is specified on the SNGCHCFLD, MLTCHCFLD, or PSHBTNFLD keyword.
6	6	1	BIT(1)	WDFKNORSTCSR	If on, *NORSTCSR parameter is specified on the SNGCHCFLD, MLTCHCFLD, or PSHBTNFLD keyword.
6	6	2	BIT(1)	*	Reserved.
6	6	3	BIT(1)	WDFKSLTIND	If on, *SLTIND parameter is specified on the SNGCHCFLD or MLTCHCFLD keyword.
6	6	4	BIT(1)	WDFKNOSLTIND	If on, *NOSLTIND parameter is specified on the SNGCHCFLD or MLTCHCFLD keyword.
6	6	5	BIT(1)	*	Reserved.
6	6	6	BIT(1)	WDFKAUTOSLT	If on, *AUTOSLT parameter is specified on the SNGCHCFLD keyword.
6	6	7	BIT(1)	WDFKAUTOSLTEN	If on, *AUTOSLTENH parameter is specified on the SNGCHCFLD keyword.
6	6	8	BIT(1)	WDFKNOAUTOSLT	If on, *NOAUTOSLT parameter is specified on the SNGCHCFLD keyword.
6	6	9	BIT(1)	WDFKHORIZ	If on, *NUMCOL or *NUMROW parameter is specified on the SNGCHCFLD, MLTCHCFLD, or PSHBTNFLD keyword.
6	6	A	BIT(1)	WDFKCOLMAJOR	If on, *NUMCOL parameter is specified on the SNGCHCFLD, MLTCHCFLD, or PSHBTNFLD keyword.
6	6	B	BIT(1)	WDFKAUTOENT	If on, *AUTOENT parameter is specified on the SNGCHCFLD keyword.
6	6	C	BIT(1)	WDFKAUTOENTNN	If on, *AUTOENTNN parameter is specified on the SNGCHCFLD keyword.
6	6	D	BIT(1)	WDFKNOAUTOENT	If on, *NOAUTOENT parameter is specified on the SNGCHCFLD keyword.
6	6	E	BIT(2)	*	Reserved.
7	7		CHAR(10)	*	Reserved.

Choice Entry Structure (QDFKCHCE)

Choice entry structure. Use this structure for a category 26 keyword that has a keyword ID of X’01’. The number of entries in this structure is defined by variable WDFKCHCCT in structure QDFKCHC. This structure is defined at variable WDFKCHCS in structure QDFKCHC. The structure is ARRAY(*).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(1)	*	Reserved.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
1	1		BIN(15)	WDFKCLEN	Length of this entry. Displacement to the subsequent choice entry.
3	3		CHAR(1)	WDFKCFLG	Flags for choice entry.
3	3	0	BIT(1)	WDFKCCTXT	On indicates choice text structure is specified; off indicates choice text is in a program-to-system field.
3	3	1	BIT(1)	WDFKCRTN	If on, return control specified is set only for a menu bar choice.
3	3	2	BIT(1)	WDFKCSPC	If on, *SPACEB parameter is specified on the CHOICE or PCHBTNCHC keyword.
3	3	3	BIT(1)	WDFKCPBC	If on, command key specified on choice.
3	3	4	BIT(4)	*	Reserved.
4	4		CHAR(2)	WDFKC	Choice number.
6	6		BIN(15)	WDFKCINX26	Index into selection string for this choice.
8	8		BIN(15)	WDFKCTXTO	Displacement to choice text. If variable WDFKCCTXT is on, this is a displacement to the choice text structure (see structure QDFKCTXT, “Choice Text Structure (QDFKCTXT)” on page 196). If WDFKCCTXT is off, this is a buffer displacement to the field containing the choice text.
10	A		BIN(15)	WDFKCTXTL	Choice text length. If choice text string is specified, this is the length of the text including trailing blanks. If choice text is a program-to-system field, this is the length of the program-to-system field.
12	C		BIN(15)	WDFKCMSGO	Displacement to CHCCTL keyword structure (see structure QDFKCMSG, “CHCCTL Keyword Structure (QDFKCMSG)” on page 196). This is set only for a selection field choice (single or multiple). 0 indicates no message is specified.
14	E		BIN(15)	WDFKCACCO	Displacement to the accelerator text structure (see structure QDFKCACC, “CHCACCEL Keyword Structure (QDFKCACC)” on page 196). 0 indicates accelerator text is not specified.
16	10		CHAR(10)	WDFKCPRCD	Name of pull-down record. Set only for a menu bar choice.
16	10		CHAR(1)	WDFKCPBCAID	AID code specified on the push button choice. X’31’ through X’3C’ indicates CA/CF01-12, X’70’ through X’7F’ indicates E00-E15, X’B1’ through X’BC’ indicates CA/CF13-24, X’BD’ indicates CLEAR, X’F1’ indicates ENTER, X’F3’ indicates HELP, X’F4’ indicates Roll Down, X’F5’ indicates Roll Up, X’F6’ indicates Print, and X’F8’ indicates Home.
17	11		CHAR(9)	*	Reserved.
26	1A		BIN(15)	WDFKCRTNO	Buffer displacement to the menu bar return field (see structure QDFKMBSEPS, “MNUBARSEP Keyword Structure (QDFKMBSEPS)” on page 197). Set only for a menu bar choice. -1 indicates no return field is specified.
28	1C		BIN(15)	WDFKCRTNL	Length of the return field.
30	1E		BIN(15)	WDFKCCTLO	Buffer displacement to the choice control field (see structure QDFKCMSG, “CHCCTL Keyword Structure (QDFKCMSG)” on page 196). This is in the output buffer. -1 indicates no control field is specified.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
32	20		BIN(15)	WDFKCCTLIO	Buffer displacement to the choice control field (see structure QDFKCMSG, “CHCCTL Keyword Structure (QDFKCMSG)”). This is in the input buffer. -1 indicates no control field is specified.
34	22		CHAR(*)	WDFKCV	Additional structures.

Choice Text Structure (QDFKCTXT)

Choice text structure. The length of the text is in the fixed choice entry string (variable WDFKCTXTL in structure QDFKCHCE). The displacement to this structure from the beginning of structure QDFKCHCE is at variable WDFKCTXTO in QDFKCHCE.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFKCMNEM	Position of the mnemonic. 0 indicates no mnemonic is specified.
2	2		CHAR(*)	WDFKCTXTT	Choice text (including trailing blanks).

CHCACCEL Keyword Structure (QDFKCACC)

CHCACCEL keyword structure. The displacement to this structure from the beginning of structure QDFKCHCE is at variable WDFKCACCO in QDFKCHCE.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFKCACCSL	Length of accelerator structure.
2	2		BIN(15)	WDFKCACCL	Length of text or variable for accelerator.
4	4		BIN(15)	WDFKCACCFO	Displacement into the output buffer for the accelerator program-to-system field. -1 indicates no accelerator program-to-system field is specified.
6	6		CHAR(*)	WDFKCACCT	Accelerator text.

CHCCTL Keyword Structure (QDFKCMSG)

CHCCTL keyword structure. The displacement to this structure from the beginning of structure QDFKCHCE is at variable WDFKCCTLO in QDFKCHCE.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKCMFLG	Message flags.
0	0	0	BIT(1)	WDFKCMIDP	On indicates the message ID is the buffer displacement to program-to-system field; off indicates the actual message ID is specified.
0	0	1	BIT(1)	WDFKCMFLP	On indicates the message file is the buffer displacement to program-to-system field; off indicates the actual message file is specified.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0	2	BIT(1)	WDFKCMLBP	On indicates the message file library is the buffer displacement to program-to-system field; off indicates the actual message library is specified.
0	0	3	BIT(5)	*	Reserved.
1	1		CHAR(7)	WDFKCMID26	Message ID.
1	1		BIN(15)	WDFKMIDO	Buffer displacement to field that contains the message ID.
3	3		CHAR(5)	*	Reserved.
8	8		CHAR(10)	WDFKCMFL	Message file name.
8	8		BIN(15)	WDFKCMFLO	Buffer displacement to field that contains the message file.
10	A		CHAR(8)	*	Reserved.
18	12		CHAR(10)	WDFKCMLB	Message file library name.
18	12		BIN(15)	WDFKCMLBO	Buffer displacement to field that contains the message library.
20	14		CHAR(8)	*	Reserved.

MNUBARSEP Keyword Structure (QDFKMBSEPS)

MNUBARSEP keyword structure. Use this structure for a category 26 keyword that has a keyword ID of X’02’. This structure is defined at variable WDFKFCEX in structure QDFKFC.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKMBSEPF	MNUBARSEP keyword flags.
0	0	0	BIT(1)	WDFKMBSCP	On indicates the color is specified in a program-to-system field; off indicates the actual color is specified. **Note:** If on, use variable WDFKMBSCO; otherwise, use variable WDFKMBSCLR.
0	0	1	BIT(1)	WDFKMBSAP	On indicates the display attribute is specified in a program-to-system field; off indicates the actual attribute is specified. **Note:** If on, use variable WDFKMBSAO; otherwise, use variable WDFKMBSATR.
0	0	2	BIT(1)	WDFKMBSHP	On indicates the character is specified in a program-to-system field; off indicates the actual character is specified. **Note:** If on, use variable WDFKMBSCHO; otherwise, use variable WDFKMBSCHR.
0	0	3	BIT(5)	*	Reserved.
1	1		BIN(15)	WDFKMBSCO	Buffer displacement to field that contains the color.
1	1		CHAR(1)	WDFKMBSCLR	Actual value for *COLOR. X’00’ indicates not specified, X’3A’ indicates BLU, X’20’ indicates GRN, X’22’ indicates WHT, X’28’ indicates RED, X’30’ indicates TRQ, X’32’ indicates YLW, and X’38’ indicates PNK.
2	2		CHAR(1)	*	Reserved.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
3	3		BIN(15)	WDFKMBSAO	Buffer displacement to field that contains the attribute.
3	3		CHAR(1)	WDFKMBSATR	Actual value for *DSPATR. Combination of two or more of these values: X'00' indicates no attribute, X'30' indicates (*DSPATR CS), X'28' indicates (*DSPATR BL), X'24' indicates (*DSPATR UL), X'22' indicates (*DSPATR HI), X'21' indicates (*DSPATR RI), and X'27' indicates (*DSPATR ND). If multiple values are specified, they are ORed together.
4	4		CHAR(1)	*	Reserved.
5	5		BIN(15)	WDFKMBSCHO	Buffer displacement to field that contains the separator character.
5	5		CHAR(1)	WDFKMBSCHR	Actual separator character. X'00' indicates not specified.
6	6		CHAR(1)	*	Reserved.

Choice Keywords Structure (QDFKCHCX)

Structure for CHCAVAIL, CHCSLT, and CHCUNAVAIL keywords. Use this structure for category 26 keywords that have keyword IDs of X'03', X'04', and X'05'. This structure is defined at variable WDFKFCEX in structure QDFKFC.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(1)	*	Reserved.
1	1		CHAR(1)	WDFKCHCCLR	Color (from the MNUBARSEP keyword).
2	2		CHAR(1)	*	Reserved.
3	3		CHAR(1)	WDFKCHCATR	Display attribute. Combination of two or more of these values: X'00' indicates no attribute, X'30' indicates (*DSPATR CS), X'28' indicates (*DSPATR BL), X'24' indicates (*DSPATR UL), X'22' indicates (*DSPATR HI), X'21' indicates (*DSPATR RI), and X'27' indicates (*DSPATR ND).
4	4		CHAR(1)	*	Reserved.

ENTFLDATR Keyword Structure (QDFKEFATR)

ENTFLDATR keyword structure. Use this structure for a category 26 keyword that has a keyword ID of X'06'. This structure is defined at variable WDFKFCEX in structure QDFKFC.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(1)	WDFKEFATRF	Entry field attribute flags.
0	0	0	BIT(1)	WDFKEFACP	On indicates the color is a program-to-system field; off indicates the actual color is specified.
0	0	1	BIT(1)	WDFKEFAAP	On indicates the attribute is a program-to-system field; off indicates the actual attribute is specified.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0	2	BIT(1)	WDFKEFACV	On indicates the cursor is visible; off indicates the cursor is invisible.
0	0	3	BIT(5)	*	Reserved.
1	1		CHAR(1)	WDFKEFACLR	Actual color. X'00' indicate no color is specified.
2	2		CHAR(1)	*	Reserved.
3	3		CHAR(1)	WDFKEFAATR	Actual attribute. X'00' indicate no attribute is specified.
4	4		CHAR(1)	*	Reserved.

FLDCSRPRG Keyword Structure (QDFKFLDCP)

FLDCSRPRG keyword structure. Use this structure for a category 26 keyword that has a keyword ID of X'07'. This structure is defined at variable WDFKFCEX in structure QDFKFC.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKFCPFLDNUM	Field number.
2	2		CHAR(1)	*	Reserved.

CNTFLD Keyword Structure (QDFKCNTFLD)

CNTFLD keyword structure. Use this structure for a category 26 keyword that has a keyword ID of X'08'. This structure is defined at variable WDFKFCEX in structure QDFKFC.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKCNTFLDNUM	Width of segment.
2	2		CHAR(1)	*	Reserved.

EDTMSK Keyword Structure (QDFKEDTMSK)

EDTMSK keyword structure. Use this structure for a category 26 keyword that has a keyword ID of X'09'. This structure is defined at variable WDFKFCEX in structure QDFKFC.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKEDMNUM	Number of segments.
2	2		CHAR(*)	WDFKEDMSEG	EDTMSK keyword segment structure (see structure QDFKEDTSEG, "EDTMSK Keyword Segment Structure (QDFKEDTSEG)").

EDTMSK Keyword Segment Structure (QDFKEDTSEG)

Segment structure for EDTMSK keyword. This structure is defined at variable WDFKEDMSEG in structure QDFKEDTMSK.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKEDMPOS	Position of segment.
2	2		BIN(15)	WDFKEDMLEN	Length of segment.

SFLCHCCTL Message Structure (QDFKSMSG)

SFLCHCCTL message structure. This structure is defined at variable wdfkfcex in structure QDFKFC.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(1)	WDFKSMFLG	Flags for SFLCHCCTL keyword.
0	0	0	BIT(1)	WDFKSMIDP	If on, message ID is buffer displacement to program-to-system field.
0	0	1	BIT(1)	WDFKSMFLP	If on, message file is buffer displacement to program-to-system field.
0	0	2	BIT(1)	WDFKSMLBP	If on, message library is buffer displacement to program-to-system field.
0	0	3	BIT(5)	*	Reserved.
1	1		CHAR(7)	WDFKSMID	Message ID.
1	1		BIN(15)	WDFKSMIDO	Output buffer displacement to the field containing the message ID.
3	3		CHAR(5)	*	Reserved.
8	8		CHAR(10)	WDFKSMFL	Message file name.
8	8		BIN(15)	WDFKSMFLO	Output buffer displacement to the field containing the message file name.
10	A		CHAR(8)	*	Reserved.
18	12		CHAR(10)	WDFKSMLB	Message library name.
18	12		BIN(15)	WDFKSMLBO	Output buffer displacement to the field containing the message library name.
20	14		CHAR(8)	*	Reserved.

Category 27 Keywords (Record-Level Grid Keywords with Parameters)

The following table shows the keyword ID that corresponds to category 27 keywords. All of these keyword IDs require a structure.

ID	Keyword
X’01’	GRDATR
X’02’	GRDCLR
X’03’	GRDBOX
X’04’	GRDLIN

Record-Level Grid Keywords with Parameters Structure (QDFKGRPR)

Record-level grid keywords with parameters. The displacement to this structure from the beginning of the appropriate section (file, record, or field) is at variable WDFFCAOF in structure QDFFCCOA.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKPCNT	Number of keywords to follow.
2	2		CHAR(*)	WDFKGRKW	Category 27 parameter entries (see structure QDFKGRWP, "Record-Level Grid Keywords (QDFKGRWP)").

Record-Level Grid Keywords (QDFKGRWP)

Record-level grid keywords. This structure is defined at variable wdfkgrkw in structure QDFKGRPR.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(1)	WDFKGPID	Keyword ID.
1	1		BIN(15)	WDFKGPIN	Index into selection table. 1 indicates keyword is not optioned.
3	3		BIN(15)	WDFKGPLN	Length of parameter for keyword.
5	5		CHAR(*)	WDFKGPEX	Extra remaining portion for keywords. Actual length is in variable WDFKMPLN in structure QDFKMRWP.

GRDATR Parameters (QDFKGRDATR)

GRDATR parameters. Use this structure for category 27 keywords that have a keyword ID of X'01'. The structure is defined at variable WDFKGPEX in structure QDFKGRWP.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFKGCLRO	Buffer displacement to program-to-system field for color.
0	0		CHAR(1)	WDFKGCLOR	Color parameter. X'FE' indicates not specified.
1	1		CHAR(1)	*	Reserved.
2	2		BIN(15)	WDFKGLNTO	Buffer displacement to program-to-system field for line type.
2	2		CHAR(1)	WDFKGLT	Line type. X'FE' indicates not specified.
3	3		CHAR(1)	*	Reserved.
4	4		CHAR(1)	WDFKGCBIT	Miscellaneous flags.
4	4	0	BIT(1)	WDFKGCLRP	If on, program-to-system field was used for color.
4	4	1	BIT(1)	WDFKGLNTP	If on, program-to-system field was used for line type.
4	4	2	BIT(6)	*	Reserved.

GRDCLR Parameters Structure (QDFKGRDCLR)

GRDCLR parameters structure. Use this structure for category 27 keywords that have a keyword ID of X'02'. The structure is defined at variable WDFKGPEX in structure QDFKGRWP. The structure is ARRAY(*).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFGCFLG	Miscellaneous flags.
0	0	0	BIT(1)	WDFKGCSRP	If on, program-to-system field was used for start-row. **Note:** If on, use variable WDFKGCSRO; otherwise, use variable WDFKGCSROW.
0	0	1	BIT(1)	WDFKGCSCP	If on, program-to-system field was used for start-column. **Note:** If on, use variable WDFKGCSCO; otherwise, use variable WDFKGCSOL.
0	0	2	BIT(1)	WDFKGCDP	If on, program-to-system field was used for depth. **Note:** If on, use variable WDFKGCDEO; otherwise, use variable WDFKGCDEP.
0	0	3	BIT(1)	WDFKGCWP	If on, program-to-system field was used for width. **Note:** If on, use variable WDFKGCWIO; otherwise, use variable WDFKGCWID.
0	0	4	BIT(1)	WDFKGCNP	If on, no parameters are specified on keyword. Clear all grid lines.
0	0	5	Bit(3)	*	Extra bits.
1	1		BIN(15)	WDFKGCSROW	Start row.
1	1		BIN(15)	WDFKGCSRO	Buffer displacement to program-to-system field for row.
3	3		BIN(15)	WDFKGCSCOL	Start column.
3	3		BIN(15)	WDFKGCSCO	Buffer displacement to program-to-system field for column.
5	5		BIN(15)	WDFKGCDEP	Depth.
5	5		BIN(15)	WDFKGCDEO	Buffer displacement to program-to-system field for depth.
7	7		BIN(15)	WDFKGCWID	Width.
7	7		BIN(15)	WDFKGCWIO	Buffer displacement to program-to-system field for width.

GRDBOX Parameters (QDFKGRDBOX)

GRDBOX parameters. Use this structure for category 27 keywords that have a keyword ID of X’03’. The structure is defined at variable WDFKGPEX in structure QDFKGRWP.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKGBBIT	Miscellaneous flags.
0	0	0	BIT(1)	WDFKGBHZP	If on, program-to-system field was used for HRZ rule *TYPE parameter. **Note:** If on, use variable WDFKGBHZO; otherwise, use variable WDFKGHZV.
0	0	1	BIT(1)	WDFKGBVTP	If on, program-to-system field was used for VRT rule *TYPE parameter. **Note:** If on, use variable WDFKGBVTO; otherwise, use variable WDFKGBVT.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0	2	BIT(1)	WDFKGBCLP	If on, program-to-system field was used for color. **Note:** If on, use variable WDFKGBCLO; otherwise, use variable WDFKGBCLR.
0	0	3	BIT(1)	WDFKGBLTP	If on, program-to-system field was used for line type. **Note:** If on, use variable WDFKGBTO; otherwise, use variable WDFKGBNT.
0	0	4	BIT(4)	*	Reserved.
1	1		CHAR(1)	WDFKGBTOB	Type of box. X'04' indicates not specified or PLAIN, X'05' indicates HRZ, X'06' indicates VRT, and X'07' indicates HRZVRT.
2	2		BIN(15)	WDFKGBHZO	Buffer displacement to program-to-system field for horizontal rule on *TYPE parameter.
2	2		BIN(15)	WDFKGHZV	Horizontal rule value. X'01' indicates not specified.
4	4		BIN(15)	WDFKGBVTO	Buffer displacement to program-to-system field for vertical rule on *TYPE parameter.
4	4		BIN(15)	WDFKGVTV	Vertical rule value. X'01' indicates not specified.
6	6		BIN(15)	WDFKGBCLO	Buffer displacement to program-to-system field for color.
6	6		CHAR(1)	WDFKGBCLR	Color parameter. X'FE' indicates not specified.
7	7		CHAR(1)	*	Reserved.
8	8		BIN(15)	WDFKGBLTO	Buffer displacement to program-to-system field for line type.
8	8		CHAR(1)	WDFKGBLNT	Line type. X'00' indicates not specified.
9	9		CHAR(1)	*	Reserved.
10	A		BIN(15)	WDFKGBCTLO	Buffer displacement to program-to-system field for *CONTROL parameter. -1 indicates not specified.
12	C		CHAR(*)	WDFKGBOXD	Array for parameters that are display-size dependent. There is one entry for each display size for the file (see structure QDFKGBOXDFM, "GRDBOX Parameter Entry Structure (QDFKGBOXDFM)").

GRDBOX Parameter Entry Structure (QDFKGBOXDFM)

GRDBOX parameter entry structure. The number of entries in this structure is defined by variable WDFFSCR in structure QDFFBASE. This structure is defined at variable WDFKGBOXD in structure QDFKGRDBOX. This structure is ARRAY(*).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(1)	WDFKGBFLG	Miscellaneous flags.
0	0	0	BIT(1)	WDFKGBSRP	If on, program-to-system field was used for start-row. **Note:** If on, use variable WDFKGBSRO; otherwise, use variable WDFKGBSROW.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0	1	BIT(1)	WDFKGBSCP	If on, program-to-system field was used for start-column. **Note:** If on, use variable WDFKGBSCO ; otherwise, use variable WDFKGBSCOL.
0	0	2	BIT(1)	WDFKGBDP	If on, program-to-system field was used for depth. **Note:** If on, use variable WDFKGBDEO; otherwise, use variable WDFKGBDEP.
0	0	3	BIT(1)	WDFKGBWP	If on, program-to-system field was used for width. **Note:** If on, use variable WDFKGBWIO; otherwise, use variable WDFKGBWID.
0	0	4	BIT(4)	*	Reserved.
1	1		BIN(15)	WDFKGBSROW	Start row,
1	1		BIN(15)	WDFKGBSRO	Buffer displacement to program-to-system field for row.
3	3		BIN(15)	WDFKGBSCOL	Start column.
3	3		BIN(15)	WDFKGBSCO	Buffer displacement to program-to-system field for column.
5	5		BIN(15)	WDFKGBDEP	Depth.
5	5		BIN(15)	WDFKGBDEO	Buffer displacement to program-to-system field for depth.
7	7		BIN(15)	WDFKGBWID	Width.
7	7		BIN(15)	WDFKGBWIO	Buffer displacement to program-to-system field for width.

GRDLIN Parameters Structure (QDFKGRDLIN)

GRDLIN parameters structure. Use this structure for category 27 keywords that have a keyword ID of X'04'. The structure is defined at variable WDFKGPEX in structure QDFKGRWP.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKGLBIT	Miscellaneous flags.
0	0	0	BIT(1)	WDFKGLIVP	If on, program-to-system field was used for interval on *TYPE parameter. **Note:** If on, use variable WDFKGLINO; otherwise, use variable WDFKGLINT.
0	0	1	BIT(1)	WDFKGLRPP	If on, program-to-system field was used for repeat on *TYPE parameter. **Note:** If on, use variable WDFKGLRPO; otherwise, use variable WDFKGLRPT.
0	0	2	BIT(1)	WDFKGLCLP	If on, program-to-system field was used for color. **Note:** If on, use variable WDFKGLCLO; otherwise, use variable WDFKGLCLR.
0	0	3	BIT(1)	WDFKGLLTP	If on, program-to-system field was used for line type. **Note:** If on, use variable WDFKGLLTO; otherwise, use variable WDFKGLLNT.
0	0	4	BIT(4)	*	Reserved.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
1	1		CHAR(1)	WDFKGLTYPL	Type of line. X’00’ indicates UPPER or not specified, X’01’ indicates LOWER, X’02’ indicates left, and X’03’ indicates RIGHT.
2	2		BIN(15)	WDFKGLINO	Buffer displacement to program-to-system field for interval.
2	2		BIN(15)	WDFKGLINT	Interval. Set to 1 if not specified.
4	4		BIN(15)	WDFKGLRPO	Buffer displacement to program-to-system field for repeat.
4	4		BIN(15)	WDFKGLRPT	Repeat. Set to 1 if not specified.
6	6		BIN(15)	WDFKGLCLO	Buffer displacement to program-to-system field for color.
6	6		CHAR(1)	WDFKGLCLR	Color parameter. X’FE’ indicates not specified.
7	7		CHAR(1)	*	Reserved.
8	8		BIN(15)	WDFKGLLTO	Buffer displacement to program-to-system field for line type.
8	8		CHAR(1)	WDFKGLLNT	Line type. X’FE’ indicates not specified.
9	9		CHAR(1)	*	Reserved.
10	A		BIN(15)	WDFKGLCTLO	Buffer displacement to program-to-system field for *CONTROL parameter. -1 indicates not specified.
12	C		CHAR(*)	WDFKGLIND	Array for parameters that are display-size dependent (see structure QDFKGLINDFM, “GRDLIN Parameter Entry Structure (QDFKGLINDFM)”).

GRDLIN Parameter Entry Structure (QDFKGLINDFM)

GRDLIN parameter entry structure. The number of entries in this structure is defined by variable WDFFSCR in structure QDFFBASE. This structure is defined at variable WDFKGLIND in structure QDFKGRDLIN. This structure is ARRAY(*).

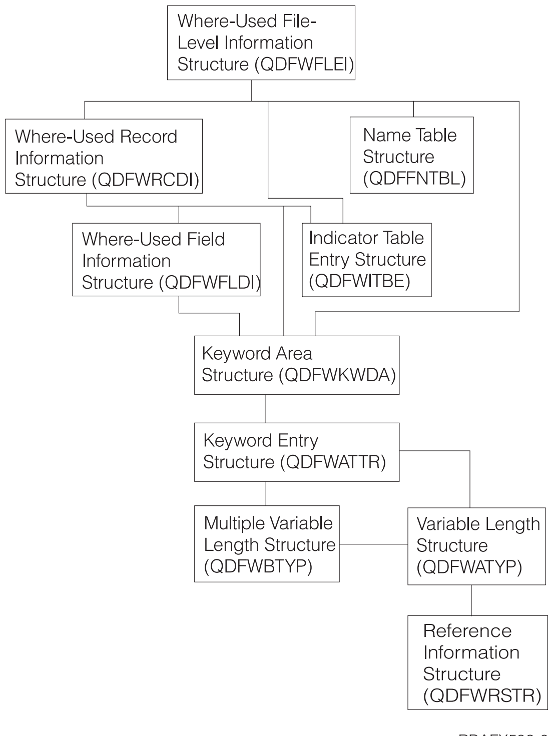
Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFKGLMISC	Miscellaneous flags.
0	0	0	BIT(1)	WDFKGLSRP	If on, program-to-system field was used for start-row. **Note:** If on, use variable WDFKGLSRO ; otherwise, use variable WDFKGLSROW.
0	0	1	BIT(1)	WDFKGLSCP	If on, program-to-system field was used for start-column. **Note:** If on, use variable WDFKGLSCO; otherwise, use variable WDFKGLSCOL.
0	0	2	BIT(1)	WDFKGLLTH	If on, program-to-system field was used for length. **Note:** If on, use variable WDFKGLLNO ; otherwise, use variable WDFKGLLEN.
0	0	3	Bit(5)	*	Reserved.
1	1		BIN(15)	WDFKGLSROW	Start row.
1	1		BIN(15)	WDFKGLSRO	Buffer displacement to program-to-system field for row.
3	3		BIN(15)	WDFKGLSCOL	Start column.
3	3		BIN(15)	WDFKGLSCO	Buffer displacement to program-to-system field for column.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
5	5		BIN(15)	WDFKGLLEN	Length.
5	5		BIN(15)	WDFKGLLNO	Buffer displacement to program-to-system field for length.

Where-Used Formats

Where-Used Section (page 206) shows the where-used section of the overview figure (DSPF0100 Format (page 135)).

Where-Used Section


Where-Used File-Level Information Structure (QDFWFLEI)
Where-Used Record Information Structure (QDFWRCDI)
Name Table Structure (QDFFNTBL)
Where-Used Field Information Structure (QDFWFLDI)
Indicator Table Entry Structure (QDFWITBE)
Keyword Area Structure (QDFWKWDA)
Keyword Entry Structure (QDFWATTR)
Multiple Variable Length Structure (QDFWBTYP)
Variable Length Structure (QDFWATYP)
Reference Information Structure (QDFWRSTR)
RBAFX593-0

Where-Used File-Level Information Structure (QDFWFLEI)

The tables in this section can be used to map to the row-column table to determine the corresponding entry in the applicable keyword table. (For the row-column table, see structure QDFFRCTB on "Row-Column Table (QDFFRCTB)" on page 151.) The where-used entries appear from left to right and top to bottom. The keyword entries appear in the same order as defined by the user.

File level information. The displacement to this structure from the beginning of structure QDFFINFO is at variable wdffwuof in QDFFINFO.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFWXLEN	Length of the file section. This is also a displacement from this structure to the first record section defined by structure QDFWRCDI ("Where-Used Record Information Structure (QDFWRCDI)" on page 207). 0 indicates internally defined files or where no record- or field-level sections exist.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
2	2		BIN(15)	WDFWXOKW	Displacement to a keyword area structure from this structure. 0, if none (see structure QDFWKWDA, "Keyword Area Structure (QDFWKWDA)" on page 209).
4	4		BIN(31)	WDFWWULN	Length of the where-used section.
8	8		BIN(31)	WDFWNTBO	Displacement from this structure to the name table defined by structure QDFFNTBL ("Name Table Structure (QDFFNTBL)" on page 211). 0 indicates the name table is not present.
12	C		BIN(15)	WDFWXIN	Number of indicator table entries (see variable WDFWINDX in this structure).
14	E		CHAR(4)	*	Reserved.
18	12		CHAR(*)	WDFWINDX	Indicator table entry structure containing the file-level indicator entries. Each entry is defined by structure QDFWITBE (page "Indicator Table Entry Structure (QDFWITBE)" on page 208).

Where-Used Record Information Structure (QDFWRCDI)

Record-level information. The displacement to this structure from the beginning of structure QDFWFLEI is at variable WDFWXLEN in QDFWFLEI. Displacements to subsequent structures are calculated using WDFWNXTR in this structure.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(15)	WDFWRLEN	Length of record-level where-used section. Also, this is the displacement from this structure to the first field if there is one in structure QDFWFLDI ("Where-Used Field Information Structure (QDFWFLDI)" on page 208).
2	2		BIN(15)	WDFWROKW	Displacement to a keyword area structure from this structure (see structure QDFWKWDA, "Keyword Area Structure (QDFWKWDA)" on page 209). 0 indicates no where-used keywords.
4	4		BIN(31)	WDFWNXTR	Length of entire where-used section for this record. This is also the displacement from this record to the next record entry.
8	8		BIN(15)	WDFWRIN	Number of indicator table entries (see variable WDFWINDR in this structure).
10	A		CHAR(2)	*	Reserved.
12	C		CHAR(*)	WDFWINDR	Indicator table containing the record-level indicator entries. These entries with the file indicator table are all the indicators (optioned and response) that are valid for this record. Each entry is defined by structure QDFWITBE ("Indicator Table Entry Structure (QDFWITBE)" on page 208).

Where-Used Field Information Structure (QDFWFLDI)

Field-level information (including constants). The displacement to this structure from the beginning of structure QDFWRCDI is at variable WDFWRLEN in QDFWRCDI. Displacements to subsequent structures are calculated using WDFWFLDL in this structure.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFWFLDL	Length of field-level where-used section. Also, this is the displacement from this structure to the next field, if there is one.
2	2		BIN(15)	WDFWFOKW	Displacement to where-used keywords from this structure. 0, if none (see structure QDFWKWDA, “Keyword Area Structure (QDFWKWDA)” on page 209).
4	4		BIN(15)	WDFWRRDX	Index into field indexing table (see structure QDFFFITB, “Field Indexing Table (QDFFFITB)” on page 152) for this field.
6	6		BIN(31)	WDFWNMEI	Index into the name table (see structure QDFFNTBL, “Name Table Structure (QDFFNTBL)” on page 211) for this field. 0 indicates constants.
10	A		BIN(15)	WDFWLFLD	Specified length of field (DDS field length). For floating-point fields, variable WDFWLFLD equals variable wdffdlen minus 7.
12	C		CHAR(1)	WDFWFFLG	Keyword flags.
12	C	0	BIT(1)	WDFWRFFD	If on, REFFLD keyword is specified.
12	C	1	BIT(1)	WDFWMGDO	If on, MSGID keyword is specified on an output-only field.
12	C	2	BIT(1)	WDFWMGDB	If on MSGID keyword is specified on a both field.
12	C	3	BIT(5)	*	Reserved.
13	D		CHAR(1)	*	Reserved.

Indicator Table Entry Structure (QDFWITBE)

Indicator table entry. These entries are obtained from the keywords and their values. This structure is defined at variable WDFWINDX in structure QDFWFLEI and variable WDFWINDR in structure QDFWRCDI.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFWINBR	Indicator number.
1	1		CHAR(1)	WDFWIOBF	Output buffer displacement. X’FF’ indicates not used as an option indicator.
2	2		CHAR(1)	WDFWIIBF	Input buffer displacement X’FF’ indicates not used as a response indicator.
3	3		BIN(15)	WDFWITXT	Indicator text displacement. File-level displacement is from structure QDFWFLEI to the indicator text for this indicator. Record-level displacement is from structure QDFWRCDI to the indicator text for this indicator. The format of the text is an A-type parameter (see structure QDFWATYP, “Variable Length Structure (QDFWATYP)” on page 210).

Keyword Area Structure (QDFWKWDA)

Keyword area. For file-level keywords, this structure is defined at variable WDFWXOKW in structure QDFWFLEI. For record-level keywords, this structure is defined at variable WDFWROKW in structure QDFWRCDI. For field-level keywords, this structure is defined at variable WDFWFOKW in structure QDFWFLDI.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFWKWDC	Keyword count.)
2	2		CHAR(*)	WDFWKWDS	Keyword entries.

Keyword Entry Structure (QDFWATTR)

Keyword entries. Keyword Types (page 209) shows the keyword types that correspond to the keyword entries and the specific structure that each keyword type uses. This figure also shows which keyword types do not require a structure.

Keyword Types

Keyword Type	Structure QDFWATYP	Structure QDFWBTYP	No Structure
ALIAS	X′001D′		
EDTWRD		X′007E′	
EDTCDE		X′007F′	
REF	X′00D8′		
REFFLD		X′00D9′	
TEXT	X′00DD′		
SFLMSGKEY			X′0187′
SFLPGMQ			X′0186′
SFLRCDNBR	X′0197′		
SFLROLVAL			X′0196′

This structure is defined at variable WDFWKWDS in structure QDFWKWDA. The structure is ARRAY(*).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFWKTYP	Keyword type (see Keyword Types (page 209)).
0	0		CHAR(1)	*	Reserved.
1	1		CHAR(1)	WDFWKWID	Where-used keyword ID.
2	2		BIN(15)	WDFWKLEN	Length of this keyword and value.
4	4		CHAR(*)	WDFWPRMS	Associated parameters. Use structure QDFWATYP (“Variable Length Structure (QDFWATYP)” on page 210) or structure QDFWBTYP (“Multiple Variable Length Structure (QDFWBTYP)” on page 210). (See Keyword Types (page 209).)

Variable Length Structure (QDFWATYP)

Variable length structure. This structure is defined at variable WDFWPRMS in structure QDFWATTR, or this structure is an array defined at variable WDFWBPRM in structure QDFWBTYP (where the number of entries is WDFWATS).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFWALEN	Length of parameter.
2	2		CHAR(1)	*	Reserved.
3	3		CHAR(1)	WDFWPRMT	Parameter type. X'00' indicates character; X'08' indicates DBCS.
4	4		CHAR(*)	WDFWAPRM	Parameter value.

Multiple Variable Length Structure (QDFWBTYP)

Multiple variable length structure. This structure is defined at variable WDFWPRMS in structure QDFWATTR.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(15)	WDFWATS	Number of variable length parameters.
2	2		CHAR(*)	WDFWBPRM	Multiple variable length structures. Each parameter is defined by structure QDFWATYP, "Variable Length Structure (QDFWATYP)."

Reference Information Structure (QDFWRSTR)

Reference information. This structure is defined at variable WDFWAPRM in structure QDFWATYP.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(1)	WDFWRFLG	Miscellaneous flags.
0	0	0	BIT(1)	WDFWSRC	If on, source reference is specified.
0	0	1	BIT(5)	*	Reserved.
0	0	6	BIT(1)	WDFWDCHK	If on, validity checking is deleted.
0	0	7	BIT(1)	WDFWDEDT	If on, editing is deleted.
1	1		CHAR(1)	WDFWRCHG	Miscellaneous flags.
1	1	0	BIT(1)	WDFWDUPE	If on, field is duplicated.
1	1	1	BIT(1)	WDFWNMEC	If on, name is changed.
1	1	2	BIT(1)	WDFWTYPC	If on, field type is changed.
1	1	3	BIT(1)	WDFWLENC	If on, field length is changed.
1	1	4	BIT(1)	WDFWDECC	If on, decimals are changed.
1	1	5	BIT(1)	WDFWEDTC	If on, editing is changed.
1	1	6	BIT(1)	WDFWVLCK	If on, validity checking is changed.
1	1	7	BIT(1)	WDFWOTHR	If on, other changes occurred.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
2	2		BIN(31)	WDFWRFEI	Index into the name table for the file name being referred to (see structure QDFFNTBL, "Name Table Structure (QDFFNTBL)").
6	6		BIN(31)	WDFWRLBI	Index into the name table for the referenced library name (see structure QDFFNTBL, "Name Table Structure (QDFFNTBL)").
10	A		BIN(31)	WDFWRRFI	Index into the name table for the referenced format name (see structure QDFFNTBL, "Name Table Structure (QDFFNTBL)").
14	E		BIN(31)	WDFWRFDI	Index into the name table for the referenced field name (see structure QDFFNTBL, "Name Table Structure (QDFFNTBL)").

Name Table Structure (QDFFNTBL)

Name table. Internally generated fields begin with *IN and end with 2 digits, such as, *IN03 and *IN27. The displacement to this structure from the beginning of structure QDFWFLEI is at variable WDFWNTBO in QDFWFLEI.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(31)	WDFFNMS	Number of names in the table.
4	4		ARRAY(*) OF CHAR(10)	WDFFNMES	Name entries.

Error Messages

Message ID	**Error Message Text**
CPF0679 E	Object &1 is not a display file.
CPF24B4 E	Severe error while addressing parameter list.
CPF3CF1 E	Error code parameter not valid.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C21 E	Format name &1 is not valid.
CPF3C22 E	Cannot get information about file &1.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V2R2

Top | "Database and File APIs," on page 1 | APIs by category

Retrieve File Override Information (QDMRTVFO) API

Required Parameter Group:

1	Returned override information	Output	Char(*)
2	Length of override information	Input	Binary(4)
3	Format name	Input	Char(8)
4	File name	Input	Char(10)
5	Error code	I/O	Char(*)

Default Public Authority: *EXCLUDE

Threadsafe: Yes

The Retrieve File Override Information (QDMRTVFO) API returns the name of the file, library, member and final type of override that result from processing TOFILE or MBR overrides for the user specified file name. Overrides will be processed in the following sequence:

- Any call level overrides up to and including the level of the activation group's oldest procedure
- Any activation group level overrides
- Any remaining call level overrides
- Overrides at the job level

Required Parameter Group

Returned override information
OUTPUT; CHAR(*)

The structure in which to return information for file overrides processed. It can be smaller than the format requested as long as the next parameter, length of override information, specifies the length correctly. When this variable is smaller than the format, the API returns only the data the variable can hold.

Length of override information
INPUT; BINARY(4)

Variable that contains the length of the user provided output area. The minimum length is 8 bytes. If you specify a length that is longer than the returned override information, the results will be unpredictable.

Format name
INPUT; CHAR(8)

The content and format of the information returned for each file. The possible format names are:

OVRL0100	File override information.

For more information, see "OVRL0100 Format" on page 213

File name
INPUT; CHAR(10)

The name of the file to return override information.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

OVRL0100 Format

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	File name used
18	12	CHAR(10)	Library name used
28	1C	CHAR(10)	Member name used
38	26	CHAR(10)	Final override type

Field Descriptions

Bytes available. The number of bytes of data available to be returned. All available data is returned if enough space is provided.

Bytes returned. The number of bytes of data returned.

File name used. The file name that results from processing file overrides for the user provided file name. This field will be set to blanks in the following cases:

- There are no overrides for the file name specified.
- Overrides exist but the TOFILE parameter was not specified.

Final override type. This field will contain the final override type applied to the file. This field will be set to the final type of override command specified as follows:

BSC	Binary synchronous communications (BSC)
CMN	Communications
DB	Database
DDM	Distributed data management
DKT	Diskette
DSP	Display
ICF	Intersystem communications function
MXD	Mixed
PRT	Printer
SAV	Save
TAP	Tape

This field will be set to blanks in the following cases:

- There are no overrides for the file name specified.
- There are overrides but neither TOFILE nor MBR parameters were specified.

Library name used. The library name which results from processing file overrides for the user provided file name including the values *LIBL and *CURLIB.

This field will be set to blanks in the following cases:
- There are no overrides for the file name specified.
- Overrides exist but TOFILE was not specified.

This will be set to *LIBL if the file name is being overridden and library name is not.

Member name used. The member name which results from processing file overrides for the user provided file name including the values *FIRST, *LAST and *ALL. This field will be set to *FIRST when the final override type is Data Base and the member is not being overridden.

This field will be set to blanks in the following cases:
- There are no overrides for the file name specified.
- Override type is not DB.
- Override type is DB, but neither TOFILE nor MBR were specified.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3CF1 E	Error code parameter not valid.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C90 E	Literal value cannot be changed.

API introduced: V3R1

Top | “Database and File APIs,” on page 1 | APIs by category

Retrieve Job Record Locks (QDBRJBRL) API

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format of receiver information	Input	Char(8)
4	Job or thread identification information	Input	Char(*)
5	Error code	I/O	Char(*)

Optional Parameter Group:

6	Format of job or thread identification information	Input	Char(8)
7	Lock filters	Input	Char(*)
8	Format of lock filters	Input	Char(8)

Default Public Authority: *USE

Threadsafe: Yes

The Retrieve Job Record Locks (QDBRJBRL) API lets you generate a list of record locks that a specific job or thread is holding or for which it is waiting. Lock information is returned for local physical files only. The Retrieve Job Record Locks API places the list in the specified receiver variable.

Authorities and Locks

Object Authority
None

Object Library Authority
None

File Lock
None

Job Authority
The API must be called from within the job for which the information is being retrieved, or the caller of the API must be running under a user profile that is the same as the job user identity of the job for which the information is being retrieved. Otherwise, the caller of the API must be running under a user profile that has job control (*JOBCTL) special authority.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)

The variable that is to receive the list of record locks. The size of this variable is specified in the Length of receiver variable parameter. See "Format of receiver information" on page 217 for details on the format of the receiver information.

Length of receiver variable
INPUT; BINARY(4)

The number of bytes that are provided in the Receiver variable parameter. At least 16 bytes must be provided. If the size of the receiver variable provided is less than the length of the list that is available, the list will be truncated; this can be determined by examining the first two fields in the receiver variable, the number of record locks returned, and the number of record locks available. If the receiver variable length specified is greater than the actual receiver variable, the results are unpredictable.

Format of receiver information
INPUT; CHAR(8)

The format of the information returned in the receiver variable. The possible format » names are «:

RJBL0100	Record lock list. See "RJBL0100 Format" on page 217 for details.
» *JOBL0100*	Record lock list. See "JOBL0100 Format" on page 218 for details. «

Job or thread identification information
INPUT; CHAR(*)

The information that is used to identify the job or thread for which the job lock information is to be returned. See "Format of job or thread identification information" on page 220 for details.

If the Format of record identification information parameter is omitted, format JIDI0100 is assumed. See "JIDI0100 Format" on page 220 for details.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Optional Parameter Group

Format of job or thread identification information
INPUT; CHAR(8)

The format of the job or thread identification information. The possible format names are:

JIDI0100	This format is used to retrieve the locks that a job is holding or waiting to hold. See "JIDI0100 Format" on page 220 for details. This is the default if this parameter is omitted.
JIDF0100	This format is used to retrieve the locks that a job or threads are holding or waiting to hold. See "JIDF0100 Format" on page 220 for details.
JIDF0200	This format is used to retrieve the locks that a specific thread is holding or waiting to hold. See "JIDF0200 Format" on page 221 for details.

Lock filters
INPUT;CHAR(*)

Filters used for the lock information that is returned. See "Format of lock filters" on page 222 for further information. If this parameter is omitted, the returned lock information is not filtered.

Format of lock filters
INPUT; CHAR(8)

The format of the lock filters used on the returned data. The possible format name is:

RJFL0100	Lock filter format. See "RJFL0100 Format" on page 222 for details.

If this parameter is omitted, the returned lock information is not filtered.

Format of receiver information

The format of the information returned in the receiver variable.

RJBL0100 Format

The following information is returned for the RJBL0100 format. For detailed descriptions of the fields in the table, see "RJBL0100 and JOBL0100 Format Field Descriptions" on page 218.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Number of record locks available
4	4	BINARY(4)	Number of record locks returned
8	8	BINARY(4)	Offset to list of record locks
12	C	BINARY(4)	Size of information for each record lock returned

Each record lock returned will have the following structure.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Database file name
10	A	CHAR(10)	Database file library name
20	14	CHAR(10)	Database member name
30	1E	CHAR(1)	Lock status
31	1F	CHAR(1)	Lock state
32	20	UNSIGNED BINARY(4)	Relative record number
36	24	CHAR(10)	Database file ASP name
46	2E	CHAR(10)	Database file library ASP name
56	38	BINARY(4)	Database file ASP number
60	3C	BINARY(4)	Database file library ASP number
64	40	CHAR(8)	Thread identifier
72	48	UNSIGNED BINARY(4)	Thread handle
76	4C	CHAR(20)	Lock space identifier
96	60	CHAR(1)	Lock scope
97	61	CHAR(3)	Reserved

»

JOBL0100 Format

The following information is returned for the JOBL0100 format. For detailed descriptions of the fields in the table, see "RJBL0100 and JOBL0100 Format Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Number of record locks available
4	4	BINARY(4)	Number of record locks returned

Each record lock returned will have the following structure.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Database file name
10	A	CHAR(10)	Database file library name
20	14	CHAR(10)	Database member name
30	1E	UNSIGNED BINARY(4)	Relative record number
34	22	CHAR(1)	Lock status

«

RJBL0100 and JOBL0100 Format Field Descriptions

Database file library name. The name of the library that contains the file.

Database file name. The name of the file.

Database member name. The name of the member.

Database file ASP name. The name of the auxiliary storage pool (ASP) that contains the file. The following special values may also be returned:

**SYSBAS*	The file is located in the system ASP or a basic user ASP.
**N*	The name of the ASP device cannot be determined.

Database file ASP number. The numeric identifier of the ASP containing the file. The following values may be returned:

1	The file is located in the system ASP.
2-32	The file is located in a basic user ASP.
33-255	The file is located in an independent ASP.
-1	The ASP number cannot be determined.

Database file library ASP name. The name of the auxiliary storage pool (ASP) that contains the library. The following special values also may be returned:

**SYSBAS*	The library is located in the system ASP or a basic user ASP.
**N*	The name of the ASP device cannot be determined.

Database file library ASP number. The numeric identifier of the ASP containing the library. The following values may be returned:

1	The library is located in the system ASP.
2-32	The library is located in a basic user ASP.
33-255	The library is located in an independent ASP.
-1	The ASP number cannot be determined.

Lock scope. The scope of the lock. The scope may be job, thread scope, or lock space. The possible values are:

0	Job scope.
1	Thread scope.
2	Lock space scope.

Lock space identifier. This field will contain a value only when the lock scope value is lock space scope and the lock is being waited on by a thread. This field will then contain the identifier of the lock space for which the lock is being waited on.

Lock state. The state of the lock. The possible values are:

0	Shared read.
1	Exclusive update.
2	Shared internal.

Lock status. The status of the lock. The possible values are:

0	The record lock is held by the given job or thread.
1	The job or thread given is waiting for the record lock.

Number of record locks available. The number of record lock structures that are available to be returned. If this field is the same as the number of record locks returned field, all the record lock information has been returned.

Number of record locks returned. The number of record lock structures that were returned to the caller of the API. If enough space is provided in the receiver variable, all record locks are returned. If there is more record lock information than can fit in the space provided, the number of record locks returned is less than the number of record locks available.

Offset to list of record locks. The byte offset from the beginning of the receiver variable to the first record lock information structure.

Relative record number. The relative record number for which record lock information is being returned.

Reserved. An unused field.

Size of information for each lock returned. The number of bytes of each of the returned lock information structures. In future releases, the amount of information returned for each lock may be expanded, so this value should be used to move from one lock structure to another.

Thread handle. This is a value which is used to address a particular thread holding a thread scope lock or the thread waiting for a lock. If the lock is a job scope lock, this is zero.

Thread identifier. The unique value that is used to identify the thread holding a thread scope lock or the thread waiting for a lock. If the lock is a job scope lock, this is zero.

Format of job or thread identification information

The format of the information needed to identify the job or thread whose locked object information is returned.

JIDI0100 Format

The following information is to be specified for the JIDI0100 format. For detailed descriptions of the fields in the table, see "JIDI0100 Format Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Job name
10	A	CHAR(10)	User name
20	14	CHAR(6)	Job number

JIDI0100 Format Field Descriptions

Job name. A specific job name.

Job number. A specific job number.

User name. A specific user profile name.

JIDF0100 Format

The following information is to be specified for the JIDF0100 format. For detailed descriptions of the fields in the table, see "JIDF0100 Format Field Descriptions" on page 221.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Job name
10	A	CHAR(10)	User name
20	14	CHAR(6)	Job number
26	1A	CHAR(16)	Internal job identifier
42	2A	CHAR(2)	Reserved
44	2C	BINARY(4)	Thread indicator
48	30	CHAR(8)	Thread identifier

JIDF0100 Format Field Descriptions

Internal job identifier. The internal identifier for the job. The List Job (QUSLJOB) API returns this identifier. If you do not specify *INT for the job name parameter, this parameter must contain blanks. With this parameter, the system can locate the job more quickly than with the job name.

Job name. A specific job name or one of the following special values:

*	The job in which this program is running. The job number and user name must contain blanks.
**INT*	The internal job identifier locates the job. The job number and user name must contain blanks.

Job number. A specific job number, or blanks when the job name specified is a special value.

Reserved. An unused field. This field must contain hexadecimal zeros.

Thread identifier. A value which is used to uniquely identify a thread within a job.

Thread indicator. The value that is used to specify the thread within the job for which information is to be retrieved. The following values are supported:

0	Information should be retrieved for the thread specified in the thread identifier field.
1	Information should be retrieved for the thread in which this program is running currently.
2	Information should be retrieved for the initial thread of the identified job.
3	Information should be retrieved for the job and its associated threads.

Note: For all supported values, the combination of the internal job identifier, job name, job number, and user name fields must identify the job containing the thread(s).

User name. A specific user profile name, or blanks when the job name specified is a special value.

JIDF0200 Format

The following information is to be specified for the JIDF0200 format. For detailed descriptions of the fields in the table, see “JIDF0200 Format Field Descriptions.”

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Job name
10	A	CHAR(10)	User name
20	14	CHAR(6)	Job number
26	1A	CHAR(16)	Internal job identifier
42	2A	CHAR(2)	Reserved
44	2C	UNSIGNED BINARY(4)	Thread handle
48	30	CHAR(8)	Thread identifier

JIDF0200 Format Field Descriptions

Internal job identifier. The internal identifier for the job. The List Job (QUSLJOB) API returns this identifier. If you do not specify *INT for the job name parameter, this parameter must contain blanks. With this parameter, the system can locate the job more quickly than with a job name.

Job name. A specific job name or one of the following special values:

*	The job in which this program is running. The job number and user name must contain blanks.
**INT*	The internal job identifier locates the job. The job number and user name must contain blanks.

Job number. A specific job number, or blanks when the job name specified is a special value.

Reserved. An unused field. This field must contain hexadecimal zeros.

Thread handle. A value which is used to address a particular thread within a job. A valid thread handle must be specified. The thread handle is returned on several other interfaces.

Thread identifier. A value which is used to uniquely identify a thread within a job.

User name. A specific user profile name, or blanks when the job name specified is a special value.

Format of lock filters

The format of the lock filters used on the returned lock information.

RJFL0100 Format

The following information is to be specified for the RJFL0100 format. For detailed descriptions of the fields in the table, see "RJFL0100 Format Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Filter size
4	4	BINARY(4)	Filter lock state
8	8	BINARY(4)	Filter lock scope
12	C	BINARY(4)	Filter lock status
16	10	CHAR(10)	Filter file name
26	1A	CHAR(10)	Filter file member name
36	24	CHAR(10)	Filter file library name
46	2E	CHAR(10)	Filter file library ASP name

RJFL0100 Format Field Descriptions

Filter lock scope: This value is used to filter information that is returned so that it contains only information about locks that have a certain lock scope.

0	Do not filter on lock scope value
1	Return only the job scope locks
2	Return only the thread scope locks
3	Return only the lock space scope locks

Default: Do not filter on lock scope value

Filter lock state: This value is used to filter information that is returned so that it contains only information about locks that have a certain lock state.

0	Do not filter on lock scope value
1	Return only the shared locks

2	Return only the exclusive locks

Default: Do not filter on lock scope value

Filter lock status: This value is used to filter information that is returned so that it contains only information about locks that have a certain lock status.

0	Do not filter on lock scope value
1	Return only locks with a status of held
2	Return only locks with a status of waiting
3	Return only locks with a status of requested.

Default: Do not filter on lock scope value

Filter file library ASP name: The name of the library's Auxiliary Storage Pool (ASP) to be filtered on. Special value of *SYSBAS can be specified. A blank field will cause no filtering to be done on this field. The default is not to filter on this field.

Filter file library name: This is the library name to be filtered on. A blank field will cause no filtering to be done on this field. The default is not to filter on this field.

Filter file member name: This is the member name to be filtered on. A blank field will cause no filtering to be done on this field. The default is not to filter on this field.

Filter file name: This is the file name to be filtered on. A blank field will cause no filtering to be done on this field. The default is not to filter on this field.

Filter size: The size of the filter information passed. Valid values are:

4	No filtering will be performed. The default values will be used for each filter.
56	All filters are required.

Error Messages

Message ID	Error Message Text
CPF0941 E	Job &3/&2/&1 no longer in system.
CPF18BF E	Thread &1 not found.
CPF1321 E	Job &1 user &2 job number &3 not found.
CPF136A E	Job &3/&2/&1 not active.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C1E E	Required parameter &1 omitted.
CPF3C21 E	Format name &1 is not valid.
CPF3C51 E	Internal job identifier not valid.
CPF3C52 E	Internal job identifier no longer valid.
CPF3C53 E	Job &3/&2/&1 not found.
CPF3C57 E	Not authorized to retrieve job information.
CPF3C58 E	Job name specified is not valid.
CPF3C59 E	Internal identifier is not blanks and job name is not *INT.
CPF3CF1 E	Error code parameter not valid.

API introduced: V5R1

Retrieve Member Description (QUSRMBRD) API

Required Parameter Group:

1	Receiver variable	
Output	Char(*)	
2	Length of receiver variable	
Input	Binary(4)	
3	Format name	
Input	Char(8)	
4	Qualified database file name	
Input	Char(20)	
5	Database member name	
Input	Char(10)	
6	Override processing	
Input	Char(1)	

Optional Parameter Group 1:

7	Error code
I/O	Char(*)

Optional Parameter Group 2:

8	Find member processing
Input	Char(1)

Default Public Authority: *USE

Threadsafe: Conditional; see “Usage Notes” on page 248.

The Retrieve Member Description (QUSRMBRD) API retrieves specific information about a single database file member and returns the information to the calling program in a receiver variable. The length of the receiver variable determines the amount of data returned. You can only use the QUSRMBRD API with database file types *PF, *LF, and *DDMF.

You can use the QUSRMBRD API to:
- Retrieve specific information about a database file member that is specified to a calling program.
- Automate reorganization when the deleted record space reaches the maximum specified.
- Ensure that the last date the source was changed matches the date the source was used to create the object.

Authorities and Locks

Library Authority
: *USE

File Authority
: *OBJOPR

File Lock
: *SHRRD

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)

The receiver variable that is to receive the information requested. You can specify that the size of the area be smaller than the format requested as long as you specify the length of the receiver variable parameter correctly. As a result, the API returns only the data the area can hold.

Length of receiver variable
INPUT; BINARY(4)

The length of the receiver variable provided. The length of receiver variable parameter may be specified up to the size of the receiver variable specified in the user program. If the length of receiver variable parameter specified is larger than the allocated size of the receiver variable specified in the user program, the results are not predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)

The content and format of the information to be returned for each specified member. The following format names are valid:

MBRD0100	*Member name and basic source information. This is similar to the information provided by the List Database File Members (QUSLMBR) API using format MBRL0200.* See "MBRD0100 Format" on page 228.
MBRD0200	Member name and expanded information. The additional information requires more system processing and takes longer to produce than the MBRD0100 format. See "MBRD0200 Format" on page 228 .
MBRD0300	Member name and full information. The additional information requires more system processing and takes longer to produce than the MBRD0200 format. See "MBRD0300 Format" on page 229.
» *MBRD0400*	Data space index information for a physical file member. See MBRD0400 Format. (page "MBRD0400 Format" on page 233) «

Qualified database file name
INPUT; CHAR(20)

The name of the database file containing the specified member whose information is to be retrieved, and the library in which it is located. The first 10 characters contain the database file name, and the second 10 characters contain the library name.

You can use these special values for the library name:

**CURLIB*	The job's current library
**LIBL*	The library list

Database member name
INPUT; CHAR(10)

The name of the database member for which information is to be retrieved. Special values follow:

**FIRST*	The first database member found.
**LAST*	The last database member found.

Override processing
INPUT; CHAR(1)

Whether overrides are to be processed. The possible values are:

0	Overrides are not processed
1	Overrides are processed

Optional Parameter Group 1

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter . If this parameter is omitted, diagnostic and escape messages are issued to the application.

Optional Parameter Group 2

Find member processing
INPUT; CHAR(1)

The method to use to find the member. There are two ways to find the member for which information is to be retrieved. The possible values are:

0	Find the file first and then look for the member in that file. This is the default value if this parameter is not specified.
1	Find the specified member directly. This method is more efficient when *LIBL is used for the library name and a specific member name is specified.

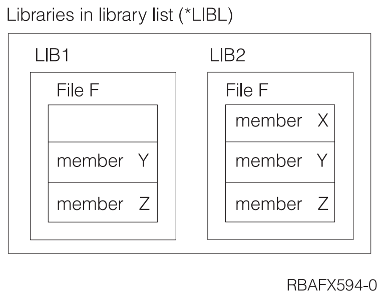
If a specific library is used to find the member, or if the member name specified is *FIRST or *LAST, the two ways will always find the same member. If *LIBL is used for the library name and a specific member name is specified (not *FIRST or *LAST), then the two ways can produce different results. See Find Member Example (page 226) .

The find member directly method is not supported when all of the following conditions exist:

- *LIBL is specified as the library.
- The member name is not specified as *FIRST or *LAST.
- The member name is not found in any of the files in the library list.
- The first occurrence of the file in the library list is a DDM file.
- The library name specified for the remote file (RMTFILE parameter on the Create DDM File (CRTDDMF) command) is *LIBL.

When this situation occurs, an error is returned from QUSRMBRD because it cannot determine which file on the remote system on OPEN operation would find. API users can monitor for this error and then re-issue the API call specifying the find file first method.

Find Member Example . File F exists in libraries LIB1 and LIB2 in the library list. If *LIBL is specified as the library for file F and member X, option 0 will not find member X because it does not exist in the file LIB1/F. Option 1 will find member X in the file LIB2/F.



Format of the Generated Information

The file member description can be provided in one of four formats:

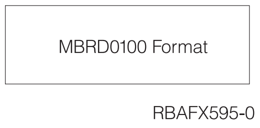
- MBRD0100
- MBRD0200
- MBRD0300
- »MBRD0400«

The structure of the information returned is determined by the value specified for the format name. For details about these formats, see the following sections. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 235.

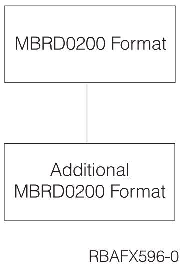
If an offset equals zero in the returned information, there is no corresponding structure associated with it.

MBRD0100 Format (page 227) , MBRD0200 Format (page 227) , and MBRD0300 Format (page 228) show how the information for the first three formats is organized. When more than one entry can appear, the figure indicates this as in **(A)** .

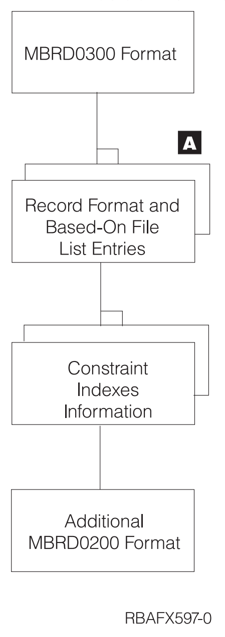
MBRD0100 Format



MBRD0200 Format



MBRD0300 Format


MBRD0300 Format
A
Record Format and Based-On File List Entries
Constraint Indexes Information
Additional MBRD0200 Format
RBAFX597-0

MBRD0100 Format

The MBRD0100 format includes the file member list and source information shown in the following table.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Database file name
18	12	CHAR(10)	Database file library name
28	1C	CHAR(10)	Member name
38	26	CHAR(10)	File attribute
48	30	CHAR(10)	Source type
58	3A	CHAR(13)	Creation date and time
71	47	CHAR(13)	» Last source change or table refresh date and time «
84	54	CHAR(50)	Member text description
134	86	CHAR(1)	Source file

MBRD0200 Format

The MBRD0200 format includes the file member name and the expanded information shown in the following table.

Offset		Type	Field
Dec	Hex		
0	0		Everything from the MBRD0100 format
135	87	CHAR(1)	Remote file
136	88	CHAR(1)	Logical file or physical file
137	89	CHAR(1)	ODP sharing
138	8A	CHAR(2)	Reserved
140	8C	BINARY(4)	Current number of records for all based-on members, if less than 2,147,483,647
144	90	BINARY(4)	Number of deleted records, if less than 2,147,483,647
148	94	BINARY(4)	Data space size
152	98	BINARY(4)	Access path size
156	9C	BINARY(4)	Number of based-on physical file members
160	A0	CHAR(13)	Change date and time
173	AD	CHAR(13)	Save date and time
186	BA	CHAR(13)	Restore date and time
199	C7	CHAR(7)	Expiration date
206	CE	CHAR(6)	Reserved
212	D4	BINARY(4)	Number of days used
216	D8	CHAR(7)	Date last used
223	DF	CHAR(7)	Use reset date
230	E6	CHAR(2)	Reserved
232	E8	BINARY(4)	Data space size multiplier
236	EC	BINARY(4)	Access path size multiplier
240	F0	BINARY(4)	Member text description CCSID
244	F4	BINARY(4)	Offset to additional MBRD0200 format information
248	F8	BINARY(4)	Length of additional MBRD0200 format information
252	FC	BINARY(4), UNSIGNED	Current number of records for all based-on members
256	100	BINARY(4), UNSIGNED	Number of deleted records
260	104	CHAR(6)	Reserved

MBRD0300 Format

The MBRD0300 format includes the file member list and the full information shown in the following table. This includes some key fields that are applicable only to the file (not member) one might use, and fields unique to the member.

Offset		Type	Field
Dec	Hex		
0	0		Everything from the MBRD0200 format
266	10A	CHAR(1)	Join member
267	10B	CHAR(1)	Access path maintenance

Offset		Type	Field
Dec	Hex		
268	10C	CHAR(10)	SQL file type
278	116	CHAR(1)	Reserved
279	117	CHAR(1)	Allow read operation
280	118	CHAR(1)	Allow write operation
281	119	CHAR(1)	Allow update operation
282	11A	CHAR(1)	Allow delete operation
283	11B	CHAR(1)	Reserved
284	11C	BINARY(4)	Records to force a write
288	120	BINARY(4)	Maximum percent deleted records allowed
292	124	BINARY(4)	Initial number of records
296	128	BINARY(4)	Increment number of records
300	12C	BINARY(4)	Maximum number of increments
304	130	BINARY(4), UNSIGNED	Current number of increments
308	134	BINARY(4), UNSIGNED	Record capacity
312	138	CHAR(10)	Record format selector program name
322	142	CHAR(10)	Record format selector library name
332	14C	BINARY(2)	Number of constraint indexes
334	14E	BINARY(4)	Offset to constraint indexes information
338	153	CHAR(46)	Reserved
384	180	Array of CHAR(112)	Record format and based-on file list
*	*	Array of CHAR(320)	Constraint indexes information

Record Format and Based-On File List Entry

The second from the last entry in the MBRD0300 format is the record format and based-on file list. There can be several entries with the information presented in the order shown in the following table. Because there can be several, it is not possible to list the exact offsets for the 112 bytes. Physical files always have only one entry. To determine the number of entries for a logical file, refer to the value in the number of based-on physical file members field in the MBRD0200 format.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Based-on physical file name
10	A	CHAR(10)	Based-on physical file library name
20	14	CHAR(10)	Based-on physical file member name
30	1E	CHAR(10)	Format name
40	28	BINARY(4)	Logical file record format number
44	2C	BINARY(4)	Current number of records, if less than 2,147,483,647
48	30	BINARY(4)	Number of deleted records, if less than 2,147,483,647
52	34	BINARY(4)	Access path size

Offset		Type	Field
Dec	Hex		
56	38	BINARY(4)	Access path size multiplier
60	3C	CHAR(1)	Access path shared
61	3D	CHAR(1)	Access path valid
62	3E	CHAR(1)	Access path held
63	3F	CHAR(10)	Access path owner file name
73	49	CHAR(10)	Access path owner library name
83	53	CHAR(10)	Access path owner member name
93	5D	CHAR(1)	Access path journaled
94	5E	CHAR(2)	Reserved
96	60	BINARY(4), UNSIGNED	Current number of records
100	64	BINARY(4), UNSIGNED	Number of deleted records
104	68	CHAR(8)	Reserved

Constraint Indexes Information

The last entry in the MBRD0300 format is the constraint indexes information list. There can be several entries with the information presented in the order shown in the following table. Because there can be several entries, it is not possible to list the exact offsets of the 321 bytes in each entry. The CHAR(8) fields (number of constraint logical-access-path read requests and the number of constraint physical-access-path read requests) are actually BINARY(8) fields and require conversion by the high-level language program that is used.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Constraint library name
10	A	BINARY(2)	Constraint name length
12	C	CHAR(258)	Constraint name
270	10E	BINARY(4)	Access path size
274	112	BINARY(4)	Access path size multiplier
278	116	CHAR(1)	Access path shared
279	117	CHAR(1)	Access path valid
280	118	CHAR(1)	Access path held
281	119	CHAR(8) [1]	Number of constraint logical-access-path read requests
289	121	CHAR(8) [1]	Number of constraint physical-access-path read requests
297	12F	CHAR(24)	Reserved
[1] CHAR(8) requires conversion to BINARY(8). Values are not supported for Version 3 Release 2.			

Additional MBRD0200 Format Information

Additional information for the MBRD0200 format is accessed using the offset to additional MBRD0200 format information and length of additional MBRD0200 format information values. The offset places the data at the end of the format requested. There can be only one entry with the information presented in the order shown in the following table. The CHAR(8) fields for the data space activity statistics and for the data space index activity statistics are actually BINARY(8) fields and require redefinition by the high-level language program used.

The counts for the data space activity statistics are intended to be approximate counts that are associated with the object since the last IPL. These counts are intended to monitor performance statistics on the object and are meant only to show trends in the operational use against the object.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(224)	Data space activity statistics
0	0	CHAR(8) [1]	Number of activate operations
8	8	CHAR(8) [1]	Number of deactivate operations
16	10	CHAR(8) [1]	Number of insert operations
24	18	CHAR(8) [1]	Number of update operations
32	20	CHAR(8) [1]	Number of delete operations
40	28	CHAR(8) [1]	Number of reset operations
48	30	CHAR(8) [1]	Number of copy operations
56	38	CHAR(8) [1]	Number of reorganize operations
64	40	CHAR(8) [1]	Number of access path build and rebuild operations
72	48	CHAR(8) [1]	Number of logical read requests
80	50	CHAR(8) [1]	Number of physical read requests
88	58	CHAR(8) [1]	Number of records rejected by key selection
96	60	CHAR(8) [1]	Number of records rejected by nonkey selection
104	68	CHAR(8) [1]	Number of records rejected by group-by selection
112	70	BINARY(4), UNSIGNED	Number of distinct valid indexes
116	74	BINARY(4), UNSIGNED	Number of distinct invalid indexes
120	78	BINARY(4), UNSIGNED	Variable length data size
» 124	7C	CHAR(1)	Rollback ended state
125	7D	CHAR(1)	Restored with partial transaction state
126	7E	CHAR(10)	*Journal receiver's name for transaction recovery*
136	88	CHAR(10)	Journal receiver's library name
146	92	CHAR(10)	Journal receiver's ASP device description name
156	9C	CHAR(36)	Reserved «
192	C0	CHAR(36)	Data space index activity statistics
192	C0	CHAR(8) [1]	Number of logical-member access-path read requests
200	C8	CHAR(8) [1]	Number of physical-member access-path read requests
208	D0	CHAR(8) [1]	Number of unique partial keys for key field 1 or number of unique full key values for an encoded vector access path. Will contain zero for access paths that do not have unique key statistics available.

Offset		Type	Field
Dec	Hex		
216	D8	CHAR(8) [1]	Number of unique partial keys for key fields 1 through 2. Will contain 0 if the access path is defined with only 1 key field, the access path is an encoded vector, or the access path does not have unique key statistics available.
224	E0	CHAR(8) [1]	Number of unique partial keys for key fields 1 through 3. Will contain 0 if the access path is defined with only less than 3 key fields, the access path is an encoded vector, or the access path does not have unique key statistics available.
232	E8	CHAR(8) [1]	Number of unique partial keys for key field 1 through 4 Will contain 0 if the access path is defined with only less than 4 key fields, the access path is an encoded vector, or the access path does not have unique key statistics available.
» 240	F0	BINARY(4), UNSIGNED	Number of overflow values
244	F4	BINARY(4), UNSIGNED	Number of delayed maintenance keys
248	F8	BINARY(4), UNSIGNED	Logical page size
252	FC	BINARY(4)	Estimated rebuild time
256	100	BINARY(2), UNSIGNED	Code size, in bytes
258	102	CHAR(13)	Last rebuild date and time
271	10F «	CHAR(13)	Reserved

[1] CHAR(8) requires redefinition to BINARY(8). Values are not supported for Version 3 Release 2.

MBRD0400 Format

The MBRD0400 format includes information about data space indexes associated with a physical file member as shown in the following table. Data space indexes are part of either file members or constraints. File members or constraints that share another file member's or constraint's data space index are not returned. If the specified member is not a physical file member, an error is returned from QUSRMBRD.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Number of data space indexes returned
12	C	BINARY(4)	Offset to the data space index list
*	*	Array of CHAR(*)	Data space index list

Data Space Index List Entry

The last entry in the MBRD0400 format is the data space index list. There can be several entries in the data space index list with the information presented in the order shown in the following table. Names in the data space index list are long alias names.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(800)	Data space index owning file member or constraint
0	0	BINARY(2)	File or constraint library name length
2	2	CHAR(258)	File or constraint library name
260	104	BINARY(2)	File or constraint name length
262	106	CHAR(258)	File or constraint name
520	208	BINARY(2)	File member name length
522	20A	CHAR(258)	File member name
780	30C	CHAR(11)	File member or constraint type
791	317	CHAR(9)	Reserved
800	320	CHAR(224)	Data space index statistics
800	320	CHAR(1)	Data space index valid
801	321	CHAR(1)	Data space index held
802	322	CHAR(6)	Reserved
808	328	CHAR(14)	Data space index creation date and time
822	336	CHAR(14)	Data space index last rebuild date and time
836	344	CHAR(14)	Data space index last query use
850	352	CHAR(14)	Data space index last query statistics use
864	360	BINARY(8)	Data space index query use count
872	368	BINARY(8)	Data space index query statistics use count
880	370	BINARY(8)	Data space index query statistics second use count
888	378	BINARY(8)	Number of data space index keys
896	380	BINARY(8)	Data space index size, in bytes
904	388	BINARY(8)	Number of unique partial keys for key field 1 or number of unique full key values for an encoded vector access path. Will contain zero for access paths that do not have unique key statistics available.
912	390	BINARY(8)	Number of unique partial keys for key fields 1 through 2. Will contain 0 if the access path is defined with only 1 key field, the access path is an encoded vector, or the access path does not have unique key statistics available.
920	398	BINARY(8)	Number of unique partial keys for key fields 1 through 3. Will contain 0 if the access path is defined with only less than 3 key fields, the access path is an encoded vector, or the access path does not have unique key statistics available.
928	3A0	BINARY(8)	Number of unique partial keys for key field 1 through 4 Will contain 0 if the access path is defined with only less than 4 key fields, the access path is an encoded vector, or the access path does not have unique key statistics available.
936	3A8	BINARY(4)	Estimated rebuild time, in seconds
940	3AC	BINARY(4)	Number of delayed maintenance keys
944	3B0	BINARY(4)	Number of overflow values
948	3B4	BINARY(4)	Code size, in bytes
952	3B8	BINARY(8)	Number of logical-member access-path read requests
960	3C0	BINARY(8)	Number of physical-member access-path read requests

Offset			
Dec	**Hex**	**Type**	**Field**
968	3C8	CHAR(56)	Reserved
1024	400	CHAR(126)	Data space index attributes
1024	400	CHAR(1)	Data space index sparse
1025	401	CHAR(1)	Data space index derived key
1026	402	CHAR(1)	Data space index partitioned
1027	403	CHAR(1)	Data space index maintenance
1028	404	CHAR(1)	Data space index recovery
1029	405	CHAR(1)	Data space index type
1030	406	CHAR(1)	Data space index unique
1031	407	CHAR(1)	Data space index sort sequence
1032	408	CHAR(10)	Sort sequence table library name
1042	412	CHAR(10)	Sort sequence table name
1052	41C	CHAR(3)	Sort sequence language ID
1055	41F	CHAR(1)	Sort sequence weight
1056	420	BINARY(4)	Logical page size
1060	424	BINARY(4)	Data space index key length
1062	426	BINARY(4)	Number of data space index key fields
1064	428	CHAR(82)	Reserved
1150	47E	BIN(2)	Data space index key field names length
1152	480	CHAR(1024)	Data space index key field names «

Field Descriptions

Access path held. Indicates if rebuild of access path is held. More information can be found in the Edit Rebuild Access Path (EDTRBDAP) command in the Control Language (CL) information. Possible values are:

blank	Not applicable unless the access path is for a join logical file or keyed file. Only indexes that are not valid can be held.
0	Access path is not held.
1	Access path is held.

Access path journaled. Whether the access path is journaled.

blank	Does not apply.
0	Access path is not journaled.
1	Access path is journaled.
2	Access path is journaled for system managed access path protection (SMAPP).

Access path maintenance. Specifies, for files with key fields or join logical files, the type of access path maintenance used for all members of the physical or logical file. The possible values are:

blank	Does not apply unless the access path is for a join logical file or a keyed file.

0	The access path is updated each time a record is changed, added, or deleted from a member. Files that require unique keys are 0.
1	The access path is updated when the member is opened with records that have been added, deleted, or changed from the member since the last time the member was opened.
2	The access path is completely rebuilt each time a file member is opened. The access path is maintained until the member is closed, then the access path is deleted.

Access path owner file name. The file name that owns the access path. This field only applies to join logical files or keyed files.

Access path owner library name. The library in which the file resides that owns the access path. This field only applies to join logical files or keyed files.

Access path owner member name. The member within the qualified file name that owns the access path. This field only applies to join logical files or keyed files.

Access path recovery. Whether the access path for the constraint, is rebuild immediately when damage to the access path is recognized.

blank	Does not apply.
0	Does not apply.
1	Access path is rebuilt *IMMED.

Access path shared. Whether an access path is shared. The possible values are:

blank	Does not apply unless the access path is for a join logical file or keyed file.
0	Access path is not shared by other files.
1	Access path is shared by other files.

Access path size. The access path size in bytes for this file member. If the file member is not keyed, the value 0 is returned. DDM files, which are not from a System/38 or iSeries system, return value 0.

Access path size multiplier. The value to multiply the access path size by to get its true size.

Access path valid. Whether the access path is valid. The possible values are:

blank	Does not apply unless the access path is for a join logical file or a keyed file.
Y	Index is valid.
N	Index is not valid and must be rebuilt.

Allow delete operation. Whether records in this file can be deleted. The possible values are:

Y	Records in this file can be deleted.
N	Records in this file cannot be deleted.

Allow read operation. Whether records in the physical file can be read. The possible values are:

Y	Records in this file can be read.
N	Records in this file cannot be read.

Allow update operation. Whether records in this file can be updated. The possible values are:

Y	Records in this file can be updated.
N	Records in this file cannot be updated.

Allow write operation. Whether records can be written to the file. The possible values are:

Y	Records can be written to this file.
N	Records cannot be written to this file.

Based-on physical file library name. The library in which the based-on physical file resides. This field is blank for a physical file.

Based-on physical file member name. The physical file member this logical file member is based on. The number of elements in this array is defined by the number of based-on physical file members field. This field is blank for a physical file.

Based-on physical file name. The name of the physical file that contains the data associated with the logical file member. This field is blank for a physical file.

Bytes available. The number of bytes of data available to be returned. All available data is returned if enough space is provided.

Bytes returned. The number of bytes of data returned.

Change date and time. The date and time this member was changed. This field is in the CYYMMDDHHMMSS format as follows:

C	Century, where 0 indicates years 19 *xx* and 1 indicates years 20 *xx* .
YY	Year
MM	Month
DD	Day
HH	Hour
MM	Minute
SS	Second

Code size. The length of the code assigned to each distinct key value of an encoded vector index. If the access path is not an encoded vector, the value 0 is returned.

Constraint library name. The name of the library containing the file to which the referential constraint applies.

Constraint name. The name of the referential constraint that controls the insertion, deletion, and update of fields that refer to like fields in a parent file.

Constraint name length. The length of the referential constraint name. The maximum length is 258 characters for delimited names and 128 characters for non-delimited names.

Creation date and time. The date and time the member was created. This field is in the CYYMMDDHHMMSS format, which is described in the change date and time field description.

Current number of increments. The number of increments that have been added to the member size (data space size). This field is 0 for logical files because the number of increments only applies to physical files.

Current number of records. The number of records that currently exist in this member. A keyed logical file member returns the number of index entries. A nonkeyed logical file member returns the number of records in the based-on physical file member. If the requested physical file member is suspended, the value 0 is returned.

Current number of records, if less than 2,147,483,647. The number of records that currently exist in this member. A keyed logical file member returns the number of index entries. A nonkeyed logical file member returns the number of records in the based-on physical file member. If the requested physical file member is suspended, the value -1 is returned. If the number of records is greater than or equal to 2,147,483,647, the value -2 is returned.

Current number of records for all based-on members. The number of records that currently exist in this member. A logical member returns the summarization of index entries. If the requested physical file member is suspended, the value 0 is returned.

Current number of records for all based-on members, if less than 2,147,483,647. The number of records that currently exist in this member. A logical member returns the summarization of index entries. If the requested physical file member is suspended, the value -1 is returned. If the number of records is greater than or equal to 2,147,483,647, the value -2 is returned.

Data space activity statistics. Information on the activity that has occurred on this member since the last IPL. All of these values are reset to binary 0 the first time the object is used after or during an IPL.

Data space index activity statistics. Information on the activity that has occurred on this member access path since the last IPL. All of these values are reset to binary 0 the first time the object is used after or during an IPL.

» **Data space index attributes.** Information on the attributes of the data space index.

Data space index creation date and time. The date and time this data space index was created. This is in the YYYYMMDDHHMMSS format.

Data space index derived key. Whether the each key field defined for the data space index has the same attributes and is directly mapped from a field in the based on physical file member, or whether some mapping or derivation is performed to produce the key values. The possible values are:

0	Data space index does not contain derived key values.
1	Data space index contains derived key values.

Data space index held. Indicates if rebuild of the data space index is held. More information can be found in the Edit Rebuild Access Path (EDTRBDAP) command in the Control Language (CL) information. Possible values are:

0	Data space index is not held.
1	Data space index is held.

Data space index key length. The maximum length of the key of the data space index.

Data space index key field names length. The length of the key field names.

Data space index key field names. A list of each key field of the data space index. Each key field name is separated by a comma and a blank. If list of key field names will be truncated if it exceeds 1024 bytes.

Data space index last query statistics use. The date and time the last time this data space index was used for statistics when optimizing a query. This is in the YYYYMMDDHHMMSS format.

Data space index last query use. The date and time the last time this data space index was used in a query full open. This is in the YYYYMMDDHHMMSS format.

Data space index last rebuild date and time. The date and time the last time this data space index was rebuilt. This is in the YYYYMMDDHHMMSS format.

Data space index list. Information about all the data space indexes build on a physical file member.

Data space index maintenance. Specifies the type of maintenance used for the data space index. The possible values are:

0	The data space index is updated each time a record is changed, added, or deleted from a member. Unique data space indexes are 0.
1	The data space index is updated when the owning member is opened. The data space index is updated with keys that have been added, deleted, or changed in the member since the last time the member was opened.
2	The data space index is completely rebuilt each time the owning member is opened. The data space index is maintained until the member is closed, then the data space index is invalidated.

Data space index owning file member or constraint. Information on the owner of the data space index. A data space index can be owned by a file member or a constraint.

Data space index partitioned. Whether the data space index contains key values from a single physical file member or whether the data space index contains key values from more than one data space. The possible values are:

0	Data space index is not partitioned.
1	Data space index is partitioned.
2	Data space index is a multi-member logical file data space index.

Data space index query use count. The number of times the index has been used in a full open of a query since the start of V5R3 or since the count has been reset by CHGOBJD.

Data space index query statistics use count. The total number of times the index has been used for statistics when optimizing a query since the start of V5R3 or since the count has been reset by CHGOBJD.

Data space index query statistics second use count. The number of times the index has been used for more expensive statistics when optimizing a query since the start of V5R3 or since the count has been reset by CHGOBJD.

Data space index recovery. Whether the data space index is rebuild immediately when damage to the access path is recognized after an abnormal IPL.

0	Data space index is rebuilt during IPL.
1	Data space index is rebuilt after IPL.
2	Data space index is not rebuilt because it is rebuild maintenance.

Data space index size. The size of the data space index.

Data space index sort sequence. Indicates whether the keys in the index use a sort sequence. The possible values are:

0	No sort sequence.
1	An alternate collating sequence is used (ALTSEQ).

2	A sort sequence is used (SRTSEQ).

Data space index sparse. Whether the data space index contains selection criteria that limits the rows which have keys in the data space index. The possible values are:

0	Data space index is not sparse.
1	Data space index is sparse. A data space index may be sparse if it is owned by a logical file that contains select/omit specifications or if it is a temporary index used for a query.

Data space index statistics. Information on the current statistics of the data space index.

Data space index type. Indicates the type of the data space index. The possible values are:

0	Binary radix tree, maximum 1TB
1	Binary radix tree, maximum 4GB
2	Encoded vector index

Data space index unique. Indicates whether the keys in the data space index are unique and if not what order is maintained for duplicate keys. The possible values are:

0	Unique. Null values are treated the same as other values (only one key with a null value is allowed).
1	Unique. Null values are not treated the same as other values (any number of keys with a null value are allowed).
2	Not unique. Duplicates are maintained first in, first out (FIFO).
3	Not unique. Duplicates are maintained last in, first out (LIFO).
4	Not unique. Duplicates are maintained first change, first out (FCFO).

Data space index valid. Whether the data space index is valid. The possible values are:

0	Data space index is not valid and must be rebuilt.
1	Data space index is valid. «

Data space size. The size of the space that contains the data of the file member, in bytes. A logical file returns a 0.

Data space size multiplier. The value to multiply the data space size by to get its true size. Typically this is 1, but for large files, the value may be greater than 1. If the data space size multiplier is greater than 1, then the value in the data space size field is not the actual size of the file.

Database file library name. The name of the library that contains the file.

Database file name. The name of the file from which the member list was retrieved.

Date last used. The century and date this member was last used. The date last used field is in the CYYMMDD format as follows:

blank	*NONE
C	Century, where 0 indicates years 19 *xx* and 1 indicates years 20 *xx* .
YY	Year
MM	Month
DD	Day

Estimated rebuild time. The estimated time, in seconds, to completely rebuild the access path. If the access path is being rebuilt currently, the value is -1. If a delayed maintenance index is being caught up currently, the value is -2. For an encoded vector index, the value is 0.

Expiration date. The date that this member expires. This is in the CYYMMDD format, which is the same format described for the date last used field description.

File attribute. The type of file found:

PF	Physical file
LF	Logical file
DDMF	Distributed data management file

» File or constraint library name length. Length of the name of the library that contains the file member or constraint that owns the data space index.

File or constraint library name. Name of the library that contains the file member or constraint that owns the data space index.

File or constraint name length. Length of the name of the file or constraint that owns the data space index.

File or constraint name. Name of the file or constraint that owns the data space index.

File member name length. Length of the name of the file member that owns the data space index. This field is zero if the data space index is owned by a constraint.

File member name. Name of the file member that owns the data space index. This field is blank if the data space index is owned by a constraint.

File member or constraint type. Type of the file member or constraint that owns the data space index. A data space index can be owned by a file member or a constraint.

FOREIGN KEY	Referential integrity constraint
INDEX	SQL index
LOGICAL	Logical file member
PHYSICAL	Physical file member
PRIMARY KEY	Primary key constraint
TEMPORARY	Temporary index
UNIQUE	Unique constraint «

Force keyed access path. Force the access path to be keyed.

0	Do not force keyed access path
1	Force a keyed access path

Format name. The definition of how data is structured in the records contained in a file. If this is a join logical file or SQL view file, the format name is only valid for the entry in the record format and based-on file member list array.

Increment number of records. The maximum number of records that are automatically added to the member when the number of records in the member is greater than the initial member size. This field applies only to physical files and is 0 for logical files.

Initial number of records. The number of records that can be written to each member of the file before the member size is automatically extended. This field applies only to physical files and is 0 for logical files.

Join member. Whether the member's logical file member combines (in one record format) fields from two or more physical file members.

0	Not a join member
1	Join member

» **Journal receiver's ASP device description name.** The name of the ASP device description that contained the journal receiver's library.

blank	*NONE

Journal receiver's library name. The name of the library that contained the journal receiver.

blank	*NONE

Journal receiver's name for transaction recovery. The earliest journal receiver that will be needed to recover using either the Apply Journaled Changes (APYJRNCHG) or the Remove Journaled Changes (RMVJRNCHG) command.

blank	*NONE «

Last rebuild date and time. The data and time of the most recent, successful rebuild or delayed maintenance catch up of the access path. This field is in the CYYMMDDHHMMSS format, which is described in the change date and time field description.

» Last source change or table refresh date and time. For source files, the date and time that this source member was last changed. For SQL materialized query tables, the date and time that the last SQL Refresh Table statement refreshed this member. If the member has never been refreshed, this field will contain hexadecimal zeroes. This field is « in the CYYMMDDHHMMSS format, which is in the same format as the change date and time field.

Logical file or physical file. Whether the file is a logical or physical file. The possible values are:

0	Member retrieved from a physical file
1	Member retrieved from a logical file

Logical file record format number. The entry number in the record format and based-on file member list. This number then corresponds to the based-on member listed in this entry. This field only applies to logical files and is 0 for a physical file.

Logical page size. The number of bytes used for the access path's logical page size. If the access path is an encoded vector, the value 0 is returned.

Maximum number of increments. The maximum number of increments automatically added to the member size. This field only applies to physical files and is 0 for a logical file.

Maximum percentage of deleted records allowed. The maximum allowed percentage of deleted records for each member in the physical file. The percentage check is made when the member is closed. If the percentage of deleted records is greater than the value shown, a message is sent to the history log. This field only applies to physical files and is 0 when either no deleted records are allowed or the file is a logical file.

Member name. The name of the member whose description is being retrieved.

Member text description. The member's text description.

Member text description CCSID. The CCSID for the member text description. The job default CCSID of the current process will be used to translate the text. For more information about CCSID, see the Globalization topic.

Number of access path build and rebuild operations. The number of access paths, both permanent and temporary, that have been built over this member since the last IPL.

Number of access path entries. The number of access path entries the physical file has for constraints.

Number of activate operations. The number of times that an open operation has been performed over this member since the last IPL.

Number of based-on physical file members. The number of database file members for the logical file member. If the member is a physical file member, the value is 0.

Number of constraint logical-access-path read requests. The number of logical read requests that have been made on keys in this constraint access path since the last IPL. This count reflects read requests issued regardless of whether a physical read request was actually performed. Rollback operations will affect this count.

Number of constraint physical-access-path read requests. The number of read requests that resulted in actual physical I/O requests on constraint keys in this member since the last IPL. Logical read requests do not necessarily result in a physical read request. Rollback operations will affect this count.

Number of copy operations. The number of times that this member has been the target of a single-entry copy instruction since the last IPL.

» Number of data space indexes. The number of data space indexes built over the physical file member.

Number of data space index keys. The number of key values in the data space index.

Number of data space index key fields. The maximum number of key fields defined in for any key in the data space index. «

Number of days used. The number of days the member has been used. If the member does not have a last-used date, the value 0 is returned.

Number of deactivate operations. The number of times that a close operation has been performed over this member since the last IPL. Note that the difference between the number of activate operations and the number of deactivate operations will indicate the number of currently active open operations over this member.

Number of delayed maintenance keys. The number of access path entries that will be processed during delayed maintenance catch up time. If the access path is an encoded vector, the value 0 is returned.

Number of delete operations. The number of records deleted from this member since the last IPL. Delete operations performed on this member as a result of the cascade referential constraint rule will affect this count. Applying journal entries that result in delete operations will affect this count. Delete operations that occur during rollback will also affect this count.

Number of deleted records. The number of deleted records returned in the file member. Keyed logical files return a 0. DDM files that are not from a System/38 or iSeries system return a 0. If the requested physical file member is suspended, the value 0 is returned.

Number of deleted records, if less than 2,147,483,647. The number of deleted records returned in the file member. Keyed logical files return a 0. DDM files that are not from a System/38 or iSeries system return a 0. If the requested physical file member is suspended, the value -1 is returned. If the number of records is greater than or equal to 2,147,483,647, the value -2 is returned.

Number of distinct invalid indexes. The number of distinct invalid indexes built over this member. This includes the index created if the file is keyed, any indexes created for dependent keyed logical files, any indexes created for dependent join logical files, any indexes created for dependent SQL indexes, and any indexes created for unique or referential constraints on the file. Access paths that share an index are not included.

Number of distinct valid indexes. The number of distinct valid indexes built over this member. This includes the index created if the file is keyed, any indexes created for dependent keyed logical files, any indexes created for dependent join logical files, any indexes created for dependent SQL indexes, and any indexes created for unique or referential constraints on the file. Access paths that share an index are not included.

Number of insert operations. The number of records inserted into this member since the last IPL. This count does not reflect records added to a member on behalf of a single entry copy instruction. Applying journal entries that result in inserts will affect this count.

Number of logical-member access-path read requests. The number of logical read requests that have been made on keys in this member access path since the last IPL. This count reflects read requests issued regardless of whether a physical read request was actually performed. Rollback operations will affect this count.

Number of logical read requests. The number of logical read requests that have been made on entries in this member since the last IPL. This count reflects read requests issued **requests** regardless of whether a physical read request was actually performed. Rollback operations will affect this count.

Number of member level constraint information array entries. The number of entries in the member level constraint information array. The maximum number of entries is 300.

Number of overflow values. The number of unique key values that do not collate in sequential order in an encoded vector. If the access path is not an encoded vector, the value 0 is returned.

Number of physical-member access-path read requests. The number of read requests that resulted in actual physical I/O requests on keys in this member since the last IPL. Logical read requests do not necessarily result in a physical read request. Rollback operations will affect this count.

Number of physical read requests. The number of read requests that resulted in actual physical I/O requests on entries in this member since the last IPL. Logical read requests do not necessarily result in a physical read request. Rollback operations will affect this count.

Number of records rejected by group-by selection. The number of records that were rejected by the selection that is associated with group-by processing on the member.

Number of records rejected by key selection. The number of records that were rejected by key record selection in open operations that are associated with the member.

Number of records rejected by nonkey selection. The number of records that were rejected by the nonkey record selection in open operations that are associated with the member.

Number of rejected entries. The number of member entries rejected by retrieve operations since the last IPL.

Number of reorganize operations. The number of times that this member has been reorganized since the last IPL.

Number of reset operations. The number of times that this member has been cleared since the last IPL. Applying journal entries that result in clear operations will affect this count.

Number of unique partial keys for key field 1. The number of unique key values considering only the first key field for keyed access paths. If the access path is an encoded vector, this number represents the number of full key distinct values. If this value is zero, then the unique key statistics are not available for this access paths. The number of unique key values are not available for access paths restored from previous releases, have keys which contain varying length character fields, or have multiple based on files.

Number of unique partial keys for key field 1 through 2. The number of unique key values from the first two key fields for keyed access paths. If this value is zero, then the unique key statistics are not available for this partial key. This number is not available for access paths restored from previous releases, containing only 1 key field, are encoded vector, have keys which contain varying length character fields, or have multiple based on files.

Number of unique partial keys for key field 1 through 3. The number of unique key values from the first three key fields for keyed access paths. If this value is zero, then the unique key statistics are not available for this partial key. This number is not available for access paths restored from previous releases, containing less than 3 key fields, are encoded vector, have keys which contain varying length character fields, or access paths with multiple based on files.

Number of unique partial keys for key field 1 through 4. The number of unique key values from the first four key fields for keyed access paths. If this value is zero, then the unique key statistics are not available for this partial key. This number is not available for access paths restored from previous releases, containing less than 4 key fields, are encoded vector, have keys which contain varying length character fields, or access paths with multiple based on files.

Number of update operations. The number of records updated in this member since the last IPL. Updates performed on the member as a result of the set null and set default referential constraint rules will affect this count. Applying journal entries that result in updates will affect this count. Update operations that occur during rollback will also affect this count.

ODP sharing. Whether the open data path (ODP) allows sharing with other programs in the same job. Possible values are:

0	ODP sharing is not allowed. A distributed data management (DDM) file that is sent to a system other than a System/38 or iSeries system returns a 0.
1	ODP sharing is allowed.

Offset to additional MBRD0200 format information. The number of bytes from the start of the MBRD0200 format to the beginning of the additional MBRD0200 format information.

» Offset to the data space index list. The number of bytes from the start of the MBRD0400 format to the beginning of the data space index list. «

Offset to member level constraint information. The number of bytes from the start of the MBRD0200 format to the beginning of the first member level constraint information array.

Record capacity. The actual number of records this member can contain. The value is calculated by multiplying the increment number of records by the maximum number of increments, and adding the initial number of records. This field only applies to a physical file and is 0 for a logical file.

Record format and based-on file list. The number of physical file members this logical file member is based on. There is a maximum of » 256 « entries. A physical file only has one entry. See "Record Format and Based-On File List Entry" on page 230 for a list of the fields contained in this list.

Record format selector library name. The library in which the record format selector program resides. This field is blank for physical files.

Record format selector program name. The name of a record format selector program that is called when the logical file member contains more than one logical record format.

The user-written selector program is called when a record is written to the database file and a record format name is not included in the high-level language (HLL) program. The selector program receives the record as input, determines the record format used, and returns it to the database. This field is blank for physical files.

Records to force a write. The number of inserted, updated, or deleted records that are processed before the records are forced into auxiliary storage. A 0 indicates that records are not forced into auxiliary storage.

Remote file. Whether the file is a remote file. Possible values are:

0	Local file
1	Remote file

Reserved. An ignored field.

Restore date and time. The date and time that the member was last restored. The restore date and time field is in the CYYMMDDHHMMSS format, which is the same as for the change date and time field. The field contains blanks if the member was never restored. DDM files that are not from a System/38 or iSeries system return blanks.

» **Restored with partial transaction state.** The data for this physical-member is currently not usable, because it was restored from media that was created using save-while-active without waiting for transaction boundaries. Physical-member data that contains partial transactions cannot be used until either the Apply Journaled Changes (APYJRNCHG) or the Remove Journaled Changes (RMVJRNCHG) command is used to apply or remove the journal changes to the member to recover the partial transactions, or the Change Journaled Object (CHGJRNOBJ) command is used to allow the member containing partial transactions to be used. See the field, Journal receiver's name for transaction recovery (page 232) , to get the name of the earliest journal receiver that will be needed to recover using the Apply Journaled Changes or Remove Journaled Changes command. The Change Journaled Object command should only be used as a last resort, since the changes in the journal receiver will not have been applied or removed completely and therefore the member will not be at a commit boundary.

0	Physical-member is not in a restored with partial transaction state.
1	Physical-member is in a restored with partial transaction state.

Rollback ended state. The data for this physical-member is currently not usable, because a rollback that was being performed against the data for the member was ended. Physical-member data that is in a rollback ended state cannot be used until either the member is restored, or the Change Journaled Object (CHGJRNOBJ) command is used to allow the member containing partial transactions to be used. The Change Journaled Object command should only be used as a last resort, since the changes in the journal receiver will not have been rolled back completely and therefore the member will not be at a commit boundary.

0	Physical-member is not in a rollback ended state.
1	Physical-member is in a rollback ended state.

Save date and time. The date and time that this member was last saved. The save date and time field is in the CYYMMDDHHMMSS format, which is the same as the change date and time field. This field contains blanks if it was never saved. DDM files that are not from a System/38 or iSeries system return blanks.

Sort sequence table library name. The library in which the sort sequence table resides that was used to order the data space index keys. The field contains blanks if a sort sequence was not used to create the data space index.

Sort sequence table name. The sort sequence table that was used to order the data space index keys. The field contains blanks if a sort sequence was not used to create the data space index.

Sort sequence language ID. The language ID that was used to identify the sort sequence table that was used to order the data space index keys. The field contains blanks if a sort sequence was not used to create the data space index or if an alternate collating sequence was used to create the data space index.

Sort sequence weight. The sort sequence weight that was used to order the data space index keys. The field contains blanks if a sort sequence was not used to create the data space index or if an alternate collating sequence was used to create the data space index. Possible values are:

0	Unique weight
1	Shared weight

Source file. Whether the file is a source file. The possible values are:

0	Data file
1	Source file

Source type. The type of source member if this is a source file.

SQL file type. The kind of SQL file type the file is. The possible values are:

blank	Not an SQL file.
TABLE	Nonkeyed physical file that contains field characteristics.
VIEW	Logical file over one or more tables or views. This SQL file type provides a subset of data in a particular table or a combination of data from more than one table or view.
INDEX	Keyed logical file over one table that is used whenever access to records in a certain order is to be requested frequently.

Use reset date. The century and date when the days-used count was last set to 0. This field is in the CYYMMDD format, which is the same as for the date last used field. If the date is not available, this field is blank.

Variable length data size. The number of pages (4096 bytes each) of variable length data in the data space.

Usage Notes

In multithreaded jobs, this API is not threadsafe and fails for DDM files of type *SNA.

Error Messages

Message ID	**Error Message Text**
CPF24B4 E	Severe error while addressing parameter list.
CPF32DE E	Cannot get information about member &3 from file &1.
CPF32DF E	Value &1 for find member parameter is not valid.
CPF3CF1 E	Error code parameter is not valid.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C20 E	Error found by program &1.
CPF3C21 E	Format name &1 is not valid.
CPF3C22 E	Cannot get information about file &1.
CPF3C23 E	Object &1 is not a database file.
CPF3C24 E	Length of the receiver variable is not valid.
CPF3C25 E	Value &1 for file override parameter is not valid.
CPF3C26 E	File &1 has no members.
CPF3C27 E	Cannot get information about member &3 from file &1.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C90 E	Literal value cannot be changed.
CPF8100 E	All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9800 E	All CPF98xx messages could be returned. xx is from 01 to FF.

API introduced: V1R3

Top | "Database and File APIs," on page 1 | APIs by category

Retrieve Record Locks (QDBRRCDL) API

Required Parameter Group:

1	Receiver variable	
Output	Char(*)	
2	Length of receiver variable	
Input	Binary(4)	
3	Format of receiver information	
Input	Char(8)	
4	Record identification information	
Input	Char(*)	
5	Member name	
Input	Char(10)	
6	Relative record number	
Input	Unsigned binary(4)	
7	Error code	
I/O	Char(*)	

Optional Parameter Group:

8	Format of record identification information
Input	Char(8)
9	Lock filters
Input	Char(*)
10	Format of lock filters
Input	Char(8)

Default Public Authority: *USE

Threadsafe: No

The Retrieve Record Locks (QDBRRCDL) API lets you generate a list of jobs, threads and lock spaces that are either waiting for or holding locks on one or more records. The Retrieve Record Locks API places the list in the specified receiver variable. Lock information is returned for records in local physical files only and file overrides are not processed.

Authorities and Locks

Object Authority
None

Object Library Authority
*EXECUTE

Object Library ASP Device Authority
*EXECUTE

File Lock
*SHRRD

Note: If the user does not have *EXECUTE authority to the object's library and *EXECUTE authority to the object library's ASP device, the user must have *JOBCTL authority.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)

The variable that is to receive the list of record locks. The size (in bytes) of this variable is specified in the length of receiver variable parameter.

See "Format of Receiver Information" on page 251 for details on the format of the receiver information.

Length of receiver variable
INPUT; BINARY(4)

The number of bytes that are provided in the Receiver variable parameter. At least 16 bytes must be provided. If the size of the receiver variable provided is less than the length of the list that is available, the list will be truncated; this can be determined by examining the first two fields in the receiver variable, the number of record locks returned, and the number of record locks available. If the receiver variable length specified is greater than the actual receiver variable, the results are unpredictable.

Format of receiver information
INPUT; CHAR(8)

The format of the information returned in the receiver variable. The possible format names are:

RRCD0100	Job record lock list. See "RRCD0100 Format" on page 251 for details.
RRCD0200	Lock holder record lock list. See "RRCD0200 Format" on page 253 for details.

Record identification information
INPUT; CHAR(*)

The information that is to be used to identify the record or records for which locks are to be retrieved. See "Format of Record Identification Information" on page 255 for details.

If the Format of record identification information parameter is omitted, format RRRC0100 is assumed. See "RRRC0100 Format" on page 255 for details.

Member name
INPUT; CHAR(10)

The name of the member in the specified file that is to be checked for record locks. This value must be blanks if RRRC0200 is specified for the format of record identification information parameter, and in that case the member value must be specified as part of the record identification information parameter. The following special value is allowed:

**FIRST*	The first member of the specified file is used.

Relative record number
INPUT; UNSIGNED BINARY(4)

The record number in the specified file and member for which lock information is to returned. This value must be 0 if RRRC0200 is specified for the format of record identification information parameter, and in that case the relative record number value must be specified as part of the record identification information parameter. The following special value is allowed:

0	Record lock information for all records in the member should be returned.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Optional Parameter Group

Format of record identification information
INPUT; CHAR(8)

The format of the record identification information. The possible format names are:

RRRC0100	This format is used to identify the file and library for which locks are to be retrieved. See “RRRC0100 Format” on page 255 for details. This is the default if this parameter is omitted.
RRRC0200	This format is used to identify the records for which locks are to be retrieved. See “RRRC0200 Format” on page 255 for details.

Lock filters
INPUT; CHAR(*)

Filters used for the lock information that is returned. See “Format of lock filters” on page 256 for further information. If this parameter is omitted, the returned lock information is not filtered.

Format of lock filters
INPUT; CHAR(8)

The format of the lock filters used on the returned data. The possible format name is:

RRFL0100	Lock filter format. See “RJFL0100 Format” on page 256 for details.

If this parameter is omitted, the returned lock information is not filtered.

Format of Receiver Information

The format of the information returned in the receiver variable.

RRCD0100 Format

The following information is returned for RRCD0100 format. When this format is used, only job and thread scope locks are returned. Lock space scope locks are not returned. Thread scope locks for all of the job’s threads are returned. For detailed descriptions of the fields in the table, see “RRCD0100 Format Field Descriptions” on page 252.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Number of record locks available
4	4	BINARY(4)	Number of record locks returned
8	8	BINARY(4)	Offset to list of record locks
12	C	BINARY(4)	Size of information for each record lock returned

Each record lock returned will have the following structure.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Job name
10	A	CHAR(10)	Job user name
20	14	CHAR(6)	Job number
26	1A	CHAR(1)	Lock status
27	1B	CHAR(1)	Lock state
28	1C	UNSIGNED BINARY(4)	Relative record number
32	20	CHAR(8)	Thread identifier
40	28	UNSIGNED BINARY(4)	Thread handle

RRCD0100 Format Field Descriptions

Job name. The simple job name of the job that issued the lock request.

Job number. The system-assigned job number of the job that issued the lock request.

Job user name. The user name under which the job that issued the lock request is run.

Lock status. The status of the lock. The possible values are:

0	The record lock is held by the given job or thread.
1	The job or thread given is waiting for the record lock.

Lock state. The lock state to be processed. The possible values are:

0	The record lock is a shared read lock.
1	The record lock is an exclusive update lock.
2	The record lock is a shared internal lock.

Number of record locks available. The number of record lock structures that are available to be returned. If this field is the same as the number of record locks returned field, all the record lock information has been returned.

Number of record locks returned. The number of record lock structures that were returned to the caller of the API. If enough space is provided in the receiver variable, all record locks are returned. If there is more record lock information than can fit in the space provided, the number of record locks returned is less than the number of record locks available.

Offset to list of record locks. The byte offset from the beginning of the receiver variable to the first record lock information structure.

Relative record number. The relative record number for which job record lock information is being returned.

Size of information for each record lock returned. The number of bytes of each of the returned record lock information structures. In future releases, the amount of information returned for each record lock may be expanded, so this value should be used to move from one record lock structure to another.

Thread handle. This is a value which is used to address a particular thread holding a thread scope lock or the thread waiting for a lock. If the lock is not a thread scope lock, this is zero.

Thread identifier. The unique value that is used to identify the thread holding a thread scope lock or the thread waiting for a lock. If the lock is not a thread scope lock, this is hex zeros.

RRCD0200 Format

The following information is returned for RRCD0200 format. For detailed descriptions of the fields in the table, see "RRCD0200 Format Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Number of record locks available
4	4	BINARY(4)	Number of record locks returned
8	8	BINARY(4)	Offset to list of record locks
12	C	BINARY(4)	Size of information for each record lock returned

Each record lock returned will have the following structure.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Job name
10	A	CHAR(10)	Job user name
20	14	CHAR(6)	Job number
26	1A	CHAR(1)	Lock status
27	1B	CHAR(1)	Lock state
28	1C	BINARY(4)	Relative record number
32	20	CHAR(8)	Thread identifier
40	28	UNSIGNED BINARY(4)	Thread handle
44	2C	CHAR(1)	Lock scope
45	2D	CHAR(1)	Holder type
46	2E	CHAR(20)	Lock space identifier
66	42	CHAR(2)	Reserved

RRCD0200 Format Field Descriptions

Holder type. If the lock status indicates the lock is held, this indicates the type of holder. If the lock status indicates the lock is not yet held, this indicates the type of holder that is waiting on the lock. This field will be the same as the lock scope field except in the case when a thread is waiting for a lock space scope lock. The possible values are:

0	Job.
1	Thread.
2	Lock space.

Job name. The simple job name of the job that issued the lock request. If the holder type is not job or thread, this is hex zeros.

Job number. The system-assigned job number of the job that issued the lock request. If the holder type is not job or thread, this is hex zeros.

Job user name. The user name under which the job that issued the lock request is run. If the holder type is not job or thread, this is hex zeros.

Lock scope. If the lock status indicates the lock is held, this indicates the type of holder. If the lock status indicates the lock is not yet held, this indicates the type of holder will hold the lock once the lock request is satisfied. This field will be the same as the holder type field except in the case when a thread is waiting for a lock space scope lock. The possible values are:

0	Job.
1	Thread.
2	Lock space.

Lock space identifier. The identifier of the lock space that holds this lock. If the lock scope is not lock space scope, this is hex zeros.

Lock status. The status of the lock. The possible values are:

0	The record lock is held. The holder may be a job, thread or lock space as indicated by the lock holder type field.
1	The record lock is being waited on. The waiter may be a job or thread as indicated by the lock holder type field.

Lock state. The lock state to be processed. The possible values are:

0	The record lock is a shared read lock.
1	The record lock is an exclusive update lock.
2	The record lock is a shared internal lock.

Number of record locks available. The number of record lock structures that are available to be returned. If this field is the same as the number of record locks returned field, all the record lock information has been returned.

Number of record locks returned. The number of record lock structures that were returned to the caller of the API. If enough space is provided in the receiver variable, all record locks are returned. If there is more record lock information than can fit in the space provided, the number of record locks returned is less than the number of record locks available.

Offset to list of record locks. The byte offset from the beginning of the receiver variable to the first record lock information structure.

Relative record number. The relative record number for which job record lock information is being returned.

Reserved. An unused field.

Size of information for each record lock returned. The number of bytes of each of the returned record lock information structures. In future releases, the amount of information returned for each record lock may be expanded, so this value should be used to move from one record lock structure to another.

Thread handle. This is a value which is used to address a particular thread holding a thread scope lock or the thread waiting for a lock. If the holder type is not thread, this is zero.

Thread identifier. The unique value that is used to identify the thread holding a thread scope lock or the thread waiting for a lock. If the holder type is not thread, this is hex zeros.

Format of Record Identification Information

The format of the information that is to be used to identify the record or records for which locks are to be retrieved. If this parameter is specified, the member and relative record number parameters are ignored, and the member and relative number specified in the format fields are used to identify the records.

RRRC0100 Format

The following information is specified for the RRRC0100 format. For detailed descriptions of the fields in the table, see "RRRC0100 Format Field Descriptions"

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	File name
10	A	CHAR(10)	Library name

RRRC0100 Format Field Descriptions

File name. The name of the file for which record locks are to be retrieved.

Library name. The name of the library where the object is located. The library is assumed to be in the name space of the thread that called the API. You can use these special values for the library name:

**CURLIB*	The current library is used to locate the object. If there is no current library, QGPL (general purpose library) is used.
**LIBL*	The library list is used to locate the object.

RRRC0200 Format

The following information is specified for the RRRC0200 format. For detailed descriptions of the fields in the table, see "RRRC0200 Format Field Descriptions" on page 256

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Record identification information size
4	4	CHAR(10)	File name
14	E	CHAR(10)	Library name
24	18	CHAR(10)	Member name
34	22	CHAR(10)	Library ASP name
44	2C	UNSIGNED BINARY(4)	Relative record number

RRRC0200 Format Field Descriptions

File name. The name of the file for which record locks are to be retrieved.

Library ASP name. The name of the auxiliary storage pool (ASP) device that contains the file's library. Special values used are:

*	Thread name space
*SYSBAS	System or basic user ASP

Library name. The name of the library where the object is located. You can use these special values for the library name:

**CURLIB*	The current library is used to locate the object. If there is no current library, QGPL (general purpose library) is used.
**LIBL*	The library list is used to locate the object.

Member name. The name of the member in the specified file that is to be checked for record locks.

The following special value is allowed:

**FIRST*	The first member of the specified file is used.

Record identification information size. The amount of data provided for the RRRC0200 format. This field must be set to 48.

Relative record number. The record number in the specified file and member for which lock information is to be returned. The following special value is allowed:

0	Record lock information for all records in the member should be returned.

Format of lock filters

The format of the lock filters used on the returned lock information.

RJFL0100 Format

The following information is to be specified for the RJFL0100 format. For detailed descriptions of the fields in the table, see "RJFL0100 Format Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Filter size
4	4	BINARY(4)	Filter lock state
8	8	BINARY(4)	Filter lock scope
12	C	BINARY(4)	Filter lock status

RJFL0100 Format Field Descriptions

Filter lock scope: This value is used to filter information that is returned so that it contains only information about locks that have a certain lock scope.

0	Do not filter on lock scope value
1	Return only the job scope locks
2	Return only the thread scope locks
3	Return only the lock space scope locks

Default: Do not filter on lock scope value

Filter lock state: This value is used to filter information that is returned so that it contains only information about locks that have a certain lock state.

0	Do not filter on lock state value
1	Return only the shared locks
2	Return only the exclusive locks

Default: Do not filter on lock state value

Filter lock status: This value is used to filter information that is returned so that it contains only information about locks that have a certain lock status.

0	Do not filter on lock status value
1	Return only locks with a status of held
2	Return only locks with a status of waiting
3	Return only locks with a status of requested.

Default: Do not filter on lock status value

Filter size: The size of the filter information passed. Valid values are:

4	No filtering will be performed. The default values will be used for each filter.
16	All filters are required

Error Messages

Message ID	**Error Message Text**
CPF24B4 E	Severe error while addressing parameter list.
CPF3130 E	Member &2 already in use.
CPF3210 E	File &1 in library &2 not correct type.
CPF3247 E	Record number &4 does not exist in member &3.
CPF3275 E	Member &3 file &1 in &2 not found.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C1E E	Required parameter &1 omitted.
CPF3C21 E	Format name &1 is not valid.
CPF3CF1 E	Error code parameter not valid.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9810 E	Library &1 not found.
CPF9812 E	File &1 in library &2 not found.

API introduced: V5R1

Top | “Database and File APIs,” on page 1 | APIs by category

Retrieve Short Name (QDBRTVSN) API

Required Parameter Group:

1	Qualified file name
Output	Char(20)
2	Long file name
Input	Char(128)
3	Length of long file name
Input	Binary(4)
4	Library name
Input	Char(10)
5	Error code
I/O	Char(*)

Default Public Authority: *USE

Threadsafe: Yes

The Retrieve Short Name(QDBRTVSN) APIallows you to get the 10-character file name of a database file by providing the long file name of the database file. The information is returned as a qualified file name.

Authorities and Locks

File Authority
*OBJOPR

Required Parameter Group

Qualified file name
OUTPUT; CHAR(20)

The short file name being retrieved and the library in which it is located. The first 10 bytes contain the file name, and the second 10 bytes contain the library name. If the input library name is *LIBL, or *CURLIB, the library name will be returned. If the 20 bytes are blanks, this means the file name could not be returned.

Long file name
INPUT; CHAR(128)

The long file name from which the short name will be retrieved.

Length of long file name
INPUT; BINARY(4)

The length of the long file name.

Library name
INPUT; CHAR(10)

The name of the library of the file. If you use one of the special values, the actual name of the library will be returned in the qualified file name parameter. You can use the following special values for the library name:

**CURLIB*	The job's current library.
**LIBL*	The library list.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	Error Message Text
CPF2207 E	Not authorized to use object &1 in library &3 type *&2.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C22 E	Cannot get information about file &1.
CPF5715 E	File &1 in library &2 not found.

API introduced: V3R7

Top | "Database and File APIs," on page 1 | APIs by category

Database APIs

The database APIs include:
- "Call Level Interface (CLI) APIs"
- "Extended Dynamic Remote SQL (EDRS) APIs" on page 262
- "Database Miscellaneous APIs" on page 321
- "Database Performance APIs" on page 372
- "Structured Query Language (SQL)APIs" on page 458

«

Top | Database and File APIs |APIs by category

Call Level Interface (CLI) APIs

The CLI APIs are:
- Allocate a statement handle (SQLAllocStmt) allocates a new statement handle and associates it with the connection specified by the connection handle.
- Allocate connection handle (SQLAllocConnect) allocates a connection handle and associated resources within the environment identified by the input environment handle.
- Allocate environment handle (SQLAllocEnv) allocates an environment handle and associated resources.
- Allocate handle (SQLAllocHandle) allocates any type of handle.
- Cancel statement (SQLCancel) attempts to end the processing of an ongoing SQL statement operation that is executing asynchronously.
- Close cursor statement (SQLCloseCursor) closes the open cursor on a statement handle.

- Column attributes (SQLColAttributes) obtains an attribute for a column of the result set, and is also used to determine the number of columns.
- Commit or roll back a transaction (SQLEndTran) commits or rolls back the current transaction in the connection.
- Connect to a data source (SQLConnect) establishes a connection to the target database.
- Connect to a data source (expanded) (SQLDriverConnect) establishes a connection to the target database, but SQLDriverConnect() uses a connection string to determine the data source name, user ID and password.
- Copy description statement (SQLCopyDesc) copies the fields of the data structure associated with the source handle to the data structure associated with the target handle.
- Describe column attributes (SQLDescribeCol) returns the result descriptor information (column name, type, precision) for the indicated column in the result set generated by a SELECT statement.
- Determine if there are more result sets (SQLMoreResults) determines whether there is more information available on the statement handle which has been associated with a stored procedure that is returning result sets.
- Disconnect from a data source (SQLDisconnect) closes the connection associated with the database connection handle.
- Execute a statement (SQLExecute) executes a statement, that was successfully prepared using SQLPrepare(), once or multiple times.
- Execute a statement directly (SQLExecDirect) executes the specified SQL statement.
- Fetch array of rows (SQLExtendedFetch) extends the function of SQLFetch() by returning a block of data containing multiple rows (called a rowset), in the form of an array, for each bound column.
- Fetch from a scrollable cursor (SQLFetchScroll) positions the cursor based on the requested orientation, then retrieves any bound columns.
- Fetch next row (SQLFetch) advances the cursor to the next row of the result set, and retrieves any bound columns.
- Free (or reset) a statement handle (SQLFreeStmt) ends processing on the statement referenced by the statement handle.
- Free a handle (SQLFreeHandle) invalidates and frees a handle.
- Free connection handle (SQLFreeConnect) invalidates and frees the connection handle.
- Free environment handle (SQLFreeEnv) invalidates and frees the environment handle.
- Get column information for a table (SQLColumns) returns a list of columns in the specified tables.
- Get cursor name (SQLGetCursorName) returns the cursor name associated with the input statement handle.
- Get data from a column (SQLGetData) retrieves data for a single column in the current row of the result set.
- Get data type information (SQLGetTypeInfo) returns information about the data types that are supported by the DBMSs associated with DB2 UDB CLI. The information is returned in an SQL result set.
- Get descriptor field (SQLGetDescField) obtains a value from a descriptor.
- Get descriptor record (SQLGetDescRec) obtains an entire record from a descriptor.
- Get functions (SQLGetFunctions) queries whether a specific function is supported.
- Get general information (SQLGetInfo) returns general information, (including supported data conversions) about the DBMS that the application is currently connected to.
- Get index and statistics information for a base table (SQLStatistics) retrieves index information for a given table.
- Get input/output parameter information for a procedure (SQLProcedureColumns) returns a list of input and output parameters associated with a procedure.
- Get list of data sources (SQLDataSources) returns a list of target databases available, one at a time.

- Get list of procedure names (SQLProcedures) returns a list of procedure names that have been registered at the server, and which match the specified search pattern.
- Get native SQL text (SQLNativeSql) is used to show how DB2 UDB CLI interprets vendor escape clauses.
- Get next parameter for which a data value is needed (SQLParamData) is used with SQLPutData() to send long data in pieces.
- Get number of parameters in an SQL statement (SQLNumParams) returns the number of parameter markers in an SQL statement.
- Get number of result columns (SQLNumResultCols) returns the number of columns in the result set associated with the input statement handle.
- Get primary key columns of a table (SQLPrimaryKeys) returns a list of column names that comprise the primary key for a table.
- Get privileges associated with a table (SQLTablePrivileges) returns a list of tables and associated privileges for each table.
- Get privileges associated with the columns of a table (SQLColumnPrivileges) returns a list of columns and associated privileges for the specified table.
- Get row count (SQLRowCount) returns the number of rows in a table affected by an UPDATE, INSERT, or DELETE statement executed against the table, or a view based on the table.
- Get special (row identifier) columns (SQLSpecialColumns) returns unique row identifier information (primary key or unique index) for a table.
- Get SQL dialect or conformance information (SQLLanguages) returns SQL dialect or conformance information.
- Get table information (SQLTables) returns a list of table names and associated information stored in the system catalogs of the connected data source.
- Get the list of foreign key columns (SQLForeignKeys) returns information about foreign keys for the specified table.
- Get the value of a connection attribute (SQLGetConnectAttr) returns the current settings for the specified connection option.
- Get the value of a statement attribute (SQLGetStmtAttr) returns the current settings of the specified statement attribute.
- Passing data value for a parameter (SQLPutData) is called following an SQLParamData() call returning SQL_NEED_DATA to supply parameter data values.
- Prepare a statement (SQLPrepare) associates an SQL statement with the input statement handle and sends the statement to the DBMS to be prepared.
- Process the next result set (SQLNextResult) determines whether there is more information available on the statement handle which has been associated with a stored procedure that is returning result sets.
- Release all environment resources (SQLReleaseEnv) invalidates and frees the environment handle.
- Retrieve error information (SQLError) returns the diagnostic information associated with the most recently called DB2 UDB CLI function for a particular statement, connection or environment handle.
- Retrieve length of a string value (SQLGetLength) is used to retrieve the length of a large object value, referenced by a large object locator that has been returned from the server (as a result of a fetch, or an SQLGetSubString() call) during the current transaction.
- Retrieve one column of a row of the result set (SQLGetCol) retrieves data for a single column in the current row of the result set.
- Retrieve portion of a string value (SQLGetSubString) is used to retrieve a portion of a large object value, referenced by a large object locator that has been returned from the server (returned by a fetch or a previous SQLGetSubString() call) during the current transaction.
- Return description of a parameter marker (SQLDescribeParam) returns the description of a parameter marker associated with a prepared SQL statement.

- Return diagnostic information (concise) (SQLGetDiagRec) returns the diagnostic information associated with the most recently called DB2 UDB CLI function for a particular statement, connection or environment handle.
- Return diagnostic information (extensible) (SQLGetDiagField) returns the diagnostic information associated with the most recently called DB2 UDB CLI function for a particular statement, connection or environment handle.
- Return starting position of string (SQLGetPosition) is used to return the starting position of one string within a LOB value (the source).
- Returns current setting of a connect option (SQLGetConnectOption) returns the current settings for the specified connection option.
- Returns current setting of a statement option (SQLGetStmtOption) returns the current settings of the specified statement option.
- Returns current setting of an environment attribute (SQLGetEnvAttr) returns the current settings for the specified environment attribute.
- Set a connection attribute (SQLSetConnectAttr) sets connection attributes for a particular connection.
- Set a descriptor field (SQLSetDescField) sets a field in a descriptor.
- Set a descriptor record (SQLSetDescRec) sets all the attributes for a descriptor record.
- Set a statement attribute (SQLSetStmtAttr) sets an attribute of a specific statement handle.
- Set connection option (SQLSetConnectOption) sets connection attributes for a particular connection.
- Set cursor name (SQLSetCursorName) associates a cursor name with the statement handle.
- Set environment attribute (SQLSetEnvAttr) sets an environment attribute for the current environment.
- Set parameter (SQLSetParam) associates (binds) an application variable to a parameter marker in an SQL statement.
- Set statement option (SQLSetStmtOption) sets an attribute of a specific statement handle.
- Specify an input array for a parameter (SQLParamOptions) provides the ability to set multiple values for each parameter set by SQLBindParameter().
- Transaction management (SQLTransact) commits or rolls back the current transaction in the connection.

«

Top | Database and File APIs | APIs by category

Extended Dynamic Remote SQL (EDRS) APIs

The EDRS APIs are:

- "Block EDRS Access (QxdaBlockEDRS) API" on page 264 (QxdaBlockEDRS) provides functions to allow client jobs to be temporarily suspended or switched to a backup server system in a client/server environment.
- "Call Program (QxdaCallProgramEDRS) API" on page 266 (QxdaCallProgramEDRS) is used to call a user-defined program on the database server system.
- "Cancel EDRS Request (QxdaCancelEDRS) API" on page 269 (QxdaCancelEDRS) cancels a previous call to the QxdaProcessExtDynEDRS or QxdaProcessImmediateEDRS APIs.
- "Check EDRS Block Status (QxdaCheckEDRSBlock) API" on page 271 (QxdaCheckEDRSBlock) returns information about the availability status of a server system.
- "Check EDRS Block Status (QxdaCheckEDRSStatus) API" on page 273 (QxdaCheckEDRSStatus) returns information about the availability status of a server system based on the provided job-suspension user data.
- "Commit EDRS Server (QxdaCommitEDRS) API" on page 275 (QxdaCommitEDRS) is used to commit transactions on the database server.

- Commit XA (QxdaXACommit) commits work performed on behalf of the transaction branch identifier.
- “Connect to EDRS Server (QxdaConnectEDRS) API” on page 281 (QxdaConnectEDRS) is used to initiate a connection to a server system.
- “Disconnect from EDRS Server (QxdaDisconnectEDRS) API” on page 288 (QxdaDisconnectEDRS) is used to end a connection to a server system.
- “Find EDRS Job (QxdaFindEDRSJob) API” on page 290 (QxdaFindEDRSJob) is used to find all jobs with user-defined data associated with the Connect to EDRS Server (QxdaConnectEDRS) API that matches the data passed to this API.
- Forget XA (QxdaXAForget) forgets about (or invalidates) a heuristically-completed transaction branch.
- Prepare XA (QxdaXAPrepare) prepares for commitment any work performed on behalf of the transaction branch identifier.
- “Process Command (QxdaProcessCommandEDRS) API” on page 300 (QxdaProcessCommand EDRS) is used to run a system command on the database server system.
- “Process Immediate SQL Statement (QxdaProcessImmediateEDRS) API” on page 302 (QxdaProcessImmediateEDRS) is used to run an SQL statement on the database server.
- “Process Remote Extended Dynamic SQL (QxdaProcessExtDynEDRS) API” on page 304 (QxdaProcessExtDynEDRS) is used to perform extended dynamic SQL operations on the database server system.
- Recover XA (QxdaXARecover) recovers a list of Transaction Branch Identifiers.
- “Roll Back EDRS Server (QxdaRollbackEDRS) API” on page 310 (QxdaRollbackEDRS) is used to roll back transactions on the database server.
- Rollback XA (QxdaXARollback) rolls back work performed on behalf of the transaction branch identifier.
- Set Connection (QxdaSetConnection) allows you to perform create, end, suspend, and find operations related to XA transactions.
- “Set XDA Options (QxdaSetOptions) API” on page 319 (QxdaSetOptions) is used to set options related to XDA.

Top | Database and File APIs | APIs by category

Block EDRS Access (QxdaBlockEDRS) API

Required Parameter Group:

1	Input structure	
Input	Char(*)	
2	Input structure format	
Input	Char(8)	
3	Error code	
I/O	Char(*)	

Service Program: QXDADBBK

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 265

The Block EDRS Access (QxdaBlockEDRS) API provides functions to allow client jobs to be temporarily suspended or switched to a backup server system in a client/server environment. This API does not physically block the system; all access must be controlled using the functions provided by the EDRS APIs.

Authorities and Locks

The user running the API must have *JOBCTL special authority.

Required Parameter Group

Input structure
I/O; CHAR(*)

The structure to pass information about the function to perform and the systems involved. For the format of this parameter, see "BLKI0100 Format."

Input structure format
INPUT; CHAR(8)

The format of the input structure being used. The possible value is:

BLKI0100	Basic structure

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

BLKI0100 Format

The following table shows the information to pass in the BLKI0100 format. For more details about the fields in this table, see "Field Descriptions" on page 265.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(1)	Function
1	1	CHAR(256)	EDRS server system name
257	101	CHAR(256)	Backup EDRS server system name
513	201	CHAR(7)	Reserved
520	208	BINARY(4)	Offset to job-suspension user data
524	20C	BINARY(4)	Length of job-suspension user data
		CHAR(*)	Job-suspension user data

Field Descriptions

Backup EDRS server system name. The name of the system that will take over as the server system when function 2 is called. This parameter is required for function 1 and must be set to blanks for other functions. The following special value is allowed:

**RESET*	This value should be specified on function 1 when switching back to the original EDRS server system.

EDRS server system name. The name of the database server system. It is required for all functions. This should always be the original server system name, even after a backup has been associated with the system.

Function. The function to perform. The possible values are:

1 - QXDA_BLOCK	Block access to the server system specified.
2 - QXDA_SWITCH_SERVER	Associate the backup server system passed to function 1 with the original server system specified.
3 - QXDA_REGISTER_JOB	Register the current job to be notified when the specified server system is blocked. A job is notified by a SIGUSR1 signal being delivered to the job.
4 - QXDA_REMOVE_JOB	Remove the job from the list of jobs to be notified of a server system block.
5 - QXDA_UNBLOCK	Allow access to the server system that was previously blocked. This function is not allowed when *RESET is specified as the backup system name to function 1.

Job-suspension user data. The data to associate with a job or a system that is used to determine which jobs on the client system should be blocked. If no job-suspension user data is supplied, all jobs connected to the specified server system will be blocked.

Length of job-suspension user data. The length of job-suspension user data supplied.

Offset to job-suspension user data. The offset from the beginning of the input structure to the job-suspension user data in the input structure, in bytes. This value must be set to 0 for functions 2 and 5, and is optional for all other functions.

Reserved. This value must be initialized to blanks.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	Error Message Text
CPF0001 E	Error found on &1 command.
CPF180C E	Function &1 not allowed.
CPF222E E	&1 special authority is required.
CPF3C1E E	Required parameter &1 omitted.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFAE14 E	Cannot allocate &1 bytes.
CPFB751 E	Parameter &1 passed not correct.
CPFB752 E	Internal error in &1 API.
CPFB75A E	Function &1 not valid while system &2 is blocked.
CPFB75B E	Function &1 not valid while system &2 is not blocked.
CPFB75C E	System name &1 is not valid.
CPFB75D E	Function &1 not allowed.
CPFB75E E	Job not removed.

API introduced: V4R4

Top | "Database and File APIs," on page 1 | APIs by category

Call Program (QxdaCallProgramEDRS) API

Required Parameter Group:

1	Connection handle
Input	Binary(4)
2	Qualified program name
Input	Char(20)
3	Number of parameters
Input	Binary(4)
4	Parameter information
Input	Char(*)
5	Error code
I/O	Char(*)

Service Program Name: QXDAEDRS

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 268

The Call Program (QxdaCallProgramEDRS) API is used to call a user-defined program on the database server system. All parameters are passed to the program by reference.

Authorities and Locks

Any program
*USE

Library of the program
*EXECUTE

Required Parameter Group

Connection handle
INPUT; BINARY(4)

The handle number of the connection on which to call the program. The connection handle must have been generated by the QxdaConnectEDRS API in the current job and activation group.

Qualified program name
INPUT; CHAR(20)

The library and name of the program to call. The special value *LIBL may be specified for the library name; however, the library list of the server job may differ from that of the client job.

Number of parameters
INPUT; BINARY(4)

The number of parameters to pass to the program.

Parameter information
INPUT; CHAR(*)

Information about each of the parameters. This should be an array of type Qxda_ParmInfo_t, with one entry for each parameter. For the format of each array element, see "Qxda_ParmInfo_t Format."

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Qxda_ParmInfo_t Format

The following table shows the structure of the Qxda_ParmInfo_t format. For more details about the fields in this table, see "Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	PTR	Parameter address
16	10	BINARY(4)	Parameter type
20	14	BINARY(4)	Parameter length
24	18	BINARY(4)	Parameter usage
28	1C	CHAR(4)	Reserved

Field Descriptions

Parameter address. The address where an input parameter exists or where an output parameter should be returned.

Parameter length. The number of bytes allocated for the parameter.

Parameter type. The type of the parameter. The possible values are:

1 - QXDA_BIN4	The parameter at the address specified is a BINARY(4) value.
2 - QXDA_CHAR	The parameter at the address specified is a character string. If the API is being called from an OS/400 application, no CCSID conversion is performed.
3 - QXDA_HEX	The parameter at the address specified is hexadecimal data and requires no conversion.
3 - QXDA_BIN2	The parameter at the address specified is a BINARY(2) value. If the API is being called from an OS/400 application, you cannot use this value.

Parameter usage. The usage of the parameter. The possible values are:

0 - QXDA_INPUT	The parameter is used for input only.
1 - QXDA_OUTPUT	The parameter is used for output only.
2 - QXDA_IN_OUT	The parameter is used for both input and output.

Reserved. Reserved field; it must be initialized to 0x00.

Usage Notes

This function may be called from the initial thread of a job only. For the OS/400 version of this API, the QXDA_CHAR and QXDA_HEX parameter types are equivalent.

Error Messages

Message ID	Error Message Text
CPF180C E	Function &1 not allowed.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFAE14 E	Cannot allocate &1 bytes.
CPFB750 E	Connection handle specified not valid.
CPFB752 E	Internal error in &1 API.
CPFB755 E	Program &1 in library &2 not found.
CPFB756 E	Rollback operation performed.
CPFB757 E	The connection is suspended.
CPFB758 E	The EDRS server system has been switched.

API introduced: V4R4

Top | "Database and File APIs," on page 1 | APIs by category

Cancel EDRS Request (QxdaCancelEDRS) API

Required Parameter Group:

1	Connection handle
Input	Binary(4)
2	Input structure
Input	Char(*)
3	Input structure format
Input	Char(8)
4	Error code
I/O	Char(*)

Service Program Name: QXDAEDRS

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 270

The Cancel EDRS Request (QxdaCancelEDRS) API is used to cancel a previous call to the QxdaProcessExtDynEDRS or QxdaProcessImmediateEDRS APIs. All parameters are passed to the program by reference.

Authorities and Locks

Job Authority

To perform a cancel operation, you must be running under a user profile that is the same as the job user identity of the job being canceled, or the issuer of the command must be running under a user profile that has job control (*JOBCTL) special authority.

The **job user identity** is the name of the user profile by which a job is known to other jobs. It is described in more detail in the Work Management book on the V5R1 Supplemental Manuals Web site.

Required Parameter Groups

Connection handle

INPUT; BINARY(4)

The handle number of the connection on which to execute the cancel request. The connection handle must have been generated by the QxdaConnectEDRS API in the current job and activation group. The connection must have been made to the same system where the qualified job name, user, and number currently is executing.

Input structure

INPUT; CHAR(*)

The structure in which to pass information about the job to cancel. For the format of this parameter, see "CDBI0100 Format" on page 270.

Input structure format

INPUT; CHAR(8)

The format of the input structure template being used. The possible value is:

CDBI0100	Basic input structure

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error code parameter.

CDBI0100 Format

The following table shows the information to pass in the CDBI0100 format. For more details about the fields in this table, see "Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(26)	Fully qualified job name

Field Descriptions

Fully qualified job name. The fully qualified name of the job to cancel. The qualified job name has three parts:

Job name	CHAR(10). The job name.
User name	CHAR(10). The user profile name for the job.
Job number	CHAR(6). The job number.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	Error Message Text
CPFAE14 E	Cannot allocate &1 bytes.
CPFB750 E	Connection handle specified not valid.
CPFB757 E	The connection is suspended.
CPFB758 E	The EDRS server system has been switched.
CPF1344 E	Not authorized to control job &1.
CPF1321 E	Job &1 user &2 job number &3 not found.

API introduced: V5R2

Top | "Database and File APIs," on page 1 | APIs by category

Check EDRS Block Status (QxdaCheckEDRSBlock) API

Required Parameter Group:

1	Receiver variable	**Output**	Char(*)
2	Length of receiver variable	**Input**	Binary(4)
3	Receiver variable format	**Input**	Char(8)
4	EDRS server system name	**Input**	Char(256)
5	Error code	**I/O**	Char(*)

Service Program: QXDADBBK

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 272

The Check EDRS Block Status (QxdaCheckEDRSBlock) API returns information about the availability status of a server system.

Authorities and Locks

None.

Required Parameter Group

Receiver variable
I/O; CHAR(*)

The structure in which to return information about the availability status of the system specified. For the format of this parameter, see "BLKO0100 Format" on page 272.

Length of receiver variable
INPUT; BINARY(4)

The number of bytes that the calling program provides for the receiver variable.

Receiver variable format
INPUT; CHAR(8)

The format of the receiver variable being used. The possible value is:

BLKO0100	Basic structure

EDRS server system name
INPUT; CHAR(256)

The name of the database server system to check.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

BLKO0100 Format

The following table shows the information to pass in the BLKO0100 format. For more details about the fields in this table, see "Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	EDRS server status
12	12	CHAR(256)	Backup server system name
268	10C	BINARY(4)	Offset to job-suspension user data
272	110	BINARY(4)	Length of job-suspension user data
		CHAR(*)	Job-suspension user data

Field Descriptions

Backup server system name. The name of the system that is acting as the backup server system. This value will be set to blanks if the EDRS system is not blocked or switched.

Bytes available. The length of the information available to the API to return, in bytes.

Bytes returned. The actual number of bytes returned to the caller of the API.

EDRS server status. The status of the server system. The possible values are:

0	QXDA_UNBLOCKED: The EDRS system is not blocked.
1	QXDA_BLOCKED: The EDRS server system is blocked.
2	QXDA_SWITCHED: The backup system is acting as the EDRS server.

Job-suspension user data. The data associated with the block.

Length of job-suspension user data. The length of job-suspension user data returned.

Offset to job-suspension user data. The offset from the beginning of the receiver variable to the the job-suspension user data, in bytes.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	Error Message Text
CPF0001 E	Error found on &1 command.
CPF3C1E E	Required parameter &1 omitted.
CPF3C90 E	Literal value cannot be changed.

Message ID	Error Message Text
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB751 E	Parameter &1 passed not correct.
CPFB752 E	Internal error in &1 API.

API introduced: V4R4

Top | “Database and File APIs,” on page 1 | APIs by category

Check EDRS Block Status (QxdaCheckEDRSStatus) API

Required Parameter Group:

1	Receiver variable	
Output	Char(*)	
2	Length of receiver variable	
Input	Binary(4)	
3	Receiver variable format	
Input	Char(8)	
4	EDRS server system name	
Input	Char(256)	
5	Job-suspension user data	
Input	Char(*)	
6	Length of job-suspension user data	
Input	Binary(4)	
7	Error code	
I/O	Char(*)	

Service Program Name: QXDADBBK

Default Public Authority: *USE

Threadsafe: Conditional; see “Usage Notes” on page 275

The Check EDRS Block Status (QxdaCheckEDRSStatus) API returns information about the availability status of a server system based on the provided job-suspension user data.

Authorities and Locks

None.

Required Parameter Group

Receiver variable
I/O; CHAR(*)

The structure in which to return information about the availability status of the system specified. For the format of this parameter, see “BLKO0100 Format” on page 274.

Length of receiver variable
INPUT; BINARY(4)

The number of bytes that the calling program provides for the receiver variable.

Receiver variable format
INPUT; CHAR(8)

The format of the receiver variable being used. The possible value is:

BLKO0100	Basic structure

EDRS server system name
INPUT; CHAR(256)

The name of the database server system to check.

Job-suspension user data
INPUT; CHAR(*)

The data to associate with a job or a system that is used to determine which jobs on the client system should be blocked. If no job-suspension user data is supplied, a status of QXDA_BLOCKED will be returned, if at least one server has this status. In that case, QXDA_UNBLOCKED will only be returned, if all matching servers have this status.

Length of job-suspension user data
INPUT; BINARY(4)

The length of the job-suspension user data supplied.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

BLKO0100 Format

The following table shows the information to pass in the BLKO0100 format. For more details about the fields in this table, see "Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	EDRS server status
12	12	CHAR(256)	Backup server system name
268	10C	BINARY(4)	Offset to job-suspension user data
272	110	BINARY(4)	Length of job-suspension user data
		CHAR(*)	Job-suspension user data

Field Descriptions

Backup server system name. The name of the system that is acting as the backup server system. This value is set to blanks if the EDRS system is not blocked or switched.

Bytes available. The length of the information available to the API to return, in bytes.

Bytes returned. The actual number of bytes returned to the caller of the API.

EDRS server status. The status of the server system. The possible values are:

0	QXDA_UNBLOCKED: The EDRS system is not blocked.
1	QXDA_BLOCKED: The EDRS server system is blocked.
2	QXDA_SWITCHED: The backup system is acting as the EDRS server.

Job-suspension user data. The data associated with the block.

Length of job-suspension user data. The length of the job-suspension user data returned.

Offset to job-suspension user data. The offset from the beginning of the receiver variable to the the job-suspension user data, in bytes.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	Error Message Text
CPF0001 E	Error found on &1 command.
CPF3C1E E	Required parameter &1 omitted.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB751 E	Parameter &1 passed not correct.
CPFB752 E	Internal error in &1 API.

API introduced: V5R1

Top | "Database and File APIs," on page 1 | APIs by category

Commit EDRS Server (QxdaCommitEDRS) API

Required Parameter Group:

1	Connection handle
Input	Binary(4)
2	Additional commit options
Input	Binary(4)
3	SQL communications area
Output	Char(136)
4	Error code
I/O	Char(*)

Service Program: QXDAEDRS

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 276

The Commit EDRS Server (QxdaCommitEDRS) API is used to commit transactions on the database server.

Authorities and Locks

None.

Required Parameter Group

Connection handle
INPUT; BINARY(4)

The handle number of the connection on which to perform the commit operation. The connection handle must have been generated by the QxdaConnectEDRS API in the current job and activation group.

Additional commit options
INPUT; BINARY(4)

The following are valid commit options:

0	QXDA_COMMIT_WORK
1	QXDA_COMMIT_WITH_HOLD

SQL communications area
OUTPUT; CHAR(136)

Returns diagnostic information. It includes the SQLCODE variable, indicating whether an error has occurred. If SQLCODE has a value of 0 after a call to this API, the function was successful.

The format of this structure is standard and is described more completely in the DB2 UDB for iSeries SQL Programming Concepts and DB2 UDB for iSeries SQL Reference books.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	Error Message Text
CPF180C E	Function &1 not allowed.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB750 E	Connection handle specified not valid.
CPFB751 E	Parameter &1 passed not correct.
CPFB752 E	Internal error in &1 API.
CPFB757 E	The connection is suspended.
CPFB758 E	The EDRS server system has been switched.

API introduced: V4R4

Top | “Database and File APIs,” on page 1 | APIs by category

Commit XA (QxdaXACommit) API

Required Parameter Group:

1	Connection handle
Input	Binary(4)
2	Transaction branch identifier
Input	Char(*)
3	Return value
Output	Binary(4)
4	Flags
Input	Binary(4)
5	Error code
I/O	Char(*)

Service Program: QXDAEDRS

Default Public Authority: *USE

Threadsafe: Conditional; see “Usage Notes” on page 279

A transaction manager calls QxdaXACommit() to commit the work associated with the transaction branch identifier. All changes that were made to resources managed by DB2 UDB for iSeries during the transaction branch are made permanent. The connection does not have to be associated with the transaction in any way.

Authorities and Locks

None.

Required Parameter Group

Connection handle
INPUT; BINARY(4)

The handle number of the connection on which to perform the XA operation. The connection handle must have been generated by the QxdaConnectEDRS API in the current job and activation group.

Transaction branch identifier
INPUT; CHAR(*)

The transaction branch identifier. This identifier is generated by the transaction manager when it starts the transaction branch. The max length is 140 bytes.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Format identifier
4	4	BINARY(4)	Length of global transaction identifier
8	8	BINARY(4)	Length of branch qualifier
12	C	CHAR(128)	Transaction branch identifier

Format identifier:	Identifies the naming format being used. If OSI CCR naming is used, the format identifier should be set to 0. If some other format is used, a the value should be greater than 0. A value of -1 means the transaction branch identifier is null.
Length of global transaction identifier:	Identifies the first of at most two contiguous components comprised by the data field. This specifies the number of bytes (1-64) starting at the first byte of the data element. This is not null-terminated.
Length of branch qualifier:	Identifies the second of at most two contiguous components comprised by the data field. This specifies the number of bytes (1-64) starting at the first byte after the global transaction identifier. This is not null-terminated.
Transaction branch identifier:	Contains the data of the transaction branch identifier

Return value
OUTPUT; BINARY(4)

The return value of the XA operation. The following values may be returned only if *TMONEPHASE(x40000000)* was set in the flags parameter:

100	[XA_RBROLLBACK]	The transaction branch was rolled back for an unspecified reason.
101	[XA_RBCOMMFAIL]	The transaction was rolled back because a communications failure occurred within the resource manager.
102	[XA_RBDEADLOCK]	The transaction was rolled back because a deadlock condition was detected within the resource manager.
103	[XA_RBINTEGRITY]	The transaction was rolled back because the resource manager detected a violation of the integrity of its resources.
104	[XA_RBOTHER]	The resource manager rolled back the transaction branch for a reason not on this list.
105	[XA_RBPROTO]	The transaction was rolled back because a protocol error occurred in the resource manager.
106	[XA_RBTIMEOUT]	The transaction was rolled back because a timeout occurred in the resource manager.
107	[XA_RBTRANSIENT]	The transaction was rolled back because a transient error was detected in the resource manager.

The following values may be returned for all flag settings.

-7	[XAER_RMFAIL]	An error occurred that makes the resource manager unavailable.

-6	[XAER_PROTO]	**xa_commit()** was not successful. Function was called in an improper context.
-5	[XAER_INVAL]	**xa_commit()** was not successful. Incorrect arguments were specified.
-4	[XAER_NOTA]	The specified xid is not known by the resource manager.
-3	[XAER_RMERR]	**xa_commit()** was not successful. The resource manager detected an error when committing the transaction branch.
-2	[XAER_ASYNC]	**xa_commit()** was not successful. The resource manager does not support asynchronous operations.
0	[XA_OK]	**xa_commit()** was successful.
4	[XA_RETRY]	The resource manager is unable to commit the transaction branch at this time. TMNOWAIT(x10000000) was set and a blocking condition exists. All resources held on behalf of *xid remain in a prepared state. The transaction manager should issue **xa_commit()** again at a later time.
5	[XA_HEURMIX]	Work on the transaction branch was partially committed and partially rolled back.
6	[XA_HEURRB]	Work on the transaction branch was heuristically rolled back.
7	[XA_HEURCOM]	Work on the transaction branch was heuristically committed.
8	[XA_HEURHAZ]	Work on the transaction branch may have been heuristically completed.

Flags INPUT; BINARY(4)

Indicator of how to perform the XA operation. The following are valid settings of flags:

TMNOWAIT:	x10000000	Do not commit the transaction if a blocking condition exists.
TMONEPHASE:	x40000000	Use the one-phase commit optimization for the specified transaction branch.
TMNOFLAGS:	x00000000	Use if no other flags are set.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB760 E	Error encountered in &2 API, return code &1.

Related Information

The QxdaXA APIs follow the NTS (transactions with transaction scoped locks) model. For more information regarding the NTS model, see XA APIs.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include qxdaedrs.h

main()
{
err_t     err;
Qxda_CDBI0100_t *istr = (Qxda_CDBI0100_t *)instruct;
Qxda_CDBO0100_t  ostr;
int ostrsize;
XID inxid;
char gtrid[8] = "TestXA";
char bqual[6] = "Test";
inxid.formatID = 0;
inxid.gtrid_length = 6;
inxid.bqual_length = 4;
int create = XDA_SQL_TXN_CREATE;
int end = XDA_SQL_TXN_END;
int timeoutval = 60;
Qsq_sqlca_t      myca;
int   stmt1len = 21;
int flags = 0;
int ret_val;
_CPYBYTES(dtaara, gtrid, inxid.gtrid_length);
_CPYBYTES(dtaara + inxid.gtrid_length, bqual, inxid.bqual_length);
_CPYBYTES(inxid.data, dtaara, inxid.gtrid_length + inxid.bqual_length);
XID *xid = &inxid;




 memset(istr, '\0', sizeof(Qxda_CDBI0100_t));


 istr->Connection_Type = 'U';
 memset(istr->Server_Name, ' ', 256);
 istr->Commitment_Control = 'S';
 _CPYBYTES(istr->Commit_Scope, "*XA                      ", 10);
 istr->Allow_Suspend = 'Y';
 memset(istr->RDB_Specified, '0', 1);
 istr->SQLDA_Cache_Size = 10;
 istr->Offset_Job_Data = sizeof(Qxda_CDBI0100_t);
 istr->Length_Job_Data = 7;
 _CPYBYTES((char *)istr + istr->Offset_Job_Data, "CONNECT", 7);
 istr->Offset_Suspend_Data = istr->Offset_Job_Data + istr->Length_Job_Data;
 istr->Length_Suspend_Data = 7;
 memset(istr->RDB_Name, ' ', 18);
 _CPYBYTES(istr->TM_Info, "TM_Name         ", 10);
 istr->LockTimeoutVal = 10;
 ostrsize = sizeof(Qxda_CDBO0100_t);
 QxdaConnectEDRS(istr, "CDBI0100", &ostr, &ostrsize,
                  "CDBO0100", &err);


 QxdaSetXaConnection(&ostr.Connection_Handle, xid, &ret_val, &create, &timeoutval, &err);

 QxdaProcessImmediateEDRS(&ostr.Connection_Handle,
              "SELECT * FROM MYTABLE", &stmt1len, &myca, &err);


 QxdaSetXaConnection(&ostr.Connection_Handle, xid, &ret_val, &end, &timeoutval, &err);


 QxdaXAPrepare(&ostr.Connection_Handle, xid, &ret_val, &flags, &err);

 QxdaXACommit(&ostr.Connection_Handle, xid, &ret_val, &flags, &err);

}
```

API introduced: V5R3

Top | "Database and File APIs," on page 1 | APIs by category

Connect to EDRS Server (QxdaConnectEDRS) API

Required Parameter Group:

1	Input structure	
Input	Char(*)	
2	Input structure format	
Input	Char(8)	
3	Receiver variable	
Output	Char(*)	
4	Length of receiver variable	
Input	Binary(4)	
5	Receiver variable format	
Input	Char(8)	
6	Error code	
I/O	Char(*)	

Service Program: QXDAEDRS

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 287

The Connect to EDRS Server (QxdaConnectEDRS) API is used to initiate a connection between the local system (requesting system), and a server system. The connection can be local, where the server system is the local system. For non-local connections, a corresponding **shadow** job is started on the server system. If the input structure format used is the basic input structure (CDBI0100), then the shadow job is swapped to run under the same user profile, using the same job description, coded character set identifier (CCSID), and job priority as the client system job. The user profile, user password, and job description must be identical on both systems for a successful connection. If the input structure format used is CDBI0200, then the shadow job will run under the specified user profile name. It will use the specified user profile's associated job description and job priority. The CCSID will be set to the CCSID of the server job field in the input structure.

For TCP/IP or UNIX domain sockets connections, a controller job must be started on the server system before calling the QxdaConnectEDRS API on the client system. The controller job can be started by using the STRTCPSVR command, specifying the *EDRSQL server.

If a TCP/IP or UNIX domain socket connection is being requested, the password level (QPWDLVL system value) of the server system must be compatible with the requesting server. If the password level is 0 or 1, or is a pre-V5R1 system, then the requesting system should be 0 or 1. Likewise, if the V5R1 server system's password level is 2 or 3, the V5R1 requesting system should also be set to 2 or 3. Failure to coordinate the password level in this fashion will prevent a successful connection, resulting in the CPI2A5A message.

Note that the CDBI0200 format cannot be used when the connection type is 'O' (Opticonnect). CDBI0200 is intended for use only with TCP/IP and UNIX socket connections. For additional restrictions that apply to OptiConnect connections, see the OptiConnect API documentation.

The connection handle returned by this API is valid only in the same job and activation group in which it was generated. A connection cannot span multiple jobs or activation groups.

If a relational database (RDB) name is specified for either the CDBI0100 or CDBI0200 format, it must be blank padded to 18 characters, the maximum length of an RDB name. If the server system does not have any active independent ASPs, the only RDB that can be connected is the *LOCAL RDB. All other RDB names will cause the CPFB752 message to be sent to the caller. The *LOCAL RDB can be determined by viewing the 'remote location' column when executing the WRKRDBDIRE command.

» If XA commitment control is specified for either the CDBI0100 or CDBI0200 format, the Transaction Manager Information field must be blank padded to 10 characters, the maximum length, and a Lock Timeout Value should be given. Specifying a *LOCAL connection type along with a *XA commit scope value will cause the CPFB754 with reason code 2 message to be sent to the caller. Likewise, specifying a N commitment control value along with a *XA commit scope value will cause the CPFB754 with reason code 3 message to be sent to the caller. XA commitment control can only be specified over TCP/IP, UNIX domain sockets, or OPTI-CONNECT. The QxdaXA* API's use the *connection* as the XA thread of control, opposed to the XA API's, which use the *thread* as the XA thread of control. «

Authorities and Locks

None.

Required Parameter Group

Input structure
INPUT; CHAR(*)

The structure in which to pass information about the connection. For the format of this parameter, see the "CDBI0100 Format" on page 283 or the "CDBI0200 Format" on page 284.

Input structure format
INPUT; CHAR(8)

The format of the input structure template being used. The possible value is:

CDBI0100	Basic input structure

CDBI0200 Basic input structure with user profile and password fields

Receiver variable
OUTPUT; CHAR(*)

The structure in which to return information about the connection. For the format of this parameter, see “CDBO0100 Format” on page 284.

Length of receiver variable
INPUT; BINARY(4)

The number of bytes that the calling program provides for the receiver variable data.

Receiver variable format
INPUT; CHAR(8)

The format of the receiver variable template being used. The possible value is:

CDBO0100 Basic receiver variable

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

CDBI0100 Format

The following table shows the information to pass in the CDBI0100 format. For more details about the fields in this table, see “Field Descriptions” on page 285.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(1)	Connection type
1	1	CHAR(1)	Commitment control
2	2	CHAR(10)	Commit scope
12	C	CHAR(1)	Allow job suspension
13	D	CHAR(256)	Server system name
269	10D	CHAR(1)	Relational database (RDB) specified
» 270	10E	CHAR(1)	SQL hexadecimal constants «
271	10F	CHAR(1)	Reserved
272	110	BINARY(4)	SQLDA cache size
276	114	BINARY(4)	Offset to job-associated user data
280	118	BINARY(4)	Length of job-associated user data
284	11C	BINARY(4)	Offset to job-suspension user data
288	120	BINARY(4)	Length of job-suspension user data
292	124	CHAR(18)	Relational database (RDB) name
» 310	136	CHAR(10)	Transaction manager information «
» 320	140	BINARY(4)	Lock timeout value «
		CHAR(*)	Job-associated user data
		CHAR(*)	Job-suspension user data

CDBI0200 Format

The following table shows the information to pass in the CDBI0200 format. For more details about the fields in this table, see “Field Descriptions” on page 285.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(1)	Connection type
1	1	CHAR(1)	Commitment control
2	2	CHAR(10)	Commit scope
12	C	CHAR(1)	Allow job suspension
13	D	CHAR(256)	Server system name
269	10D	CHAR(1)	Convert Endian Data
270	10E	CHAR(1)	Relational database (RDB) specified
» 271	10F	CHAR(1)	SQL hexadecimal constants «
272	110	BINARY(4)	SQLDA cache size
276	114	BINARY(4)	Offset to job-associated user data
280	118	BINARY(4)	Length of job-associated user data
284	11C	BINARY(4)	Offset to job-suspension user data
288	120	BINARY(4)	Length of job-suspension user data
292	124	BINARY(4)	Offset to user profile data
296	128	BINARY(4)	Length of user profile data
300	12C	BINARY(4)	Offset to password associated with user profile
304	130	BINARY(4)	Length of password associated with user profile
308	134	BINARY(4)	CCSID of the server job
312	138	BINARY(4)	CCSID of the password
316	13C	CHAR(18)	Relational database (RDB) name
» 334	14E	CHAR(10)	Transaction manager information «
» 344	158	BINARY(4)	Lock timeout value «
		CHAR(*)	Job-associated user data
		CHAR(*)	Job-suspension user data
		CHAR(*)	User profile data
		CHAR(*)	Password associated with user profile data

CDBO0100 Format

The following table shows the information returned in the CDBO0100 format. For more details about the fields in the following table, see “Field Descriptions” on page 285.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Connection handle
12	C	CHAR(10)	Server job name

Offset			
Dec	Hex	Type	Field
22	16	CHAR(10)	Server job user name
32	20	CHAR(6)	Server job number
38	26	CHAR(1)	Connection type used

Field Descriptions

Allow job suspension. Whether or not to allow this job to be suspended or switched to run to a backup server if there is a server system failure or backup. The possible values are:

Y	This job may be suspended or switched for server system failures or backups. If this option is specified with a remote connection type and the server system has been switched to a backup by the QxdaBlockEDRS API, the connection will be made to the backup system.
N	This job should not be suspended or switched.

Bytes available. The length of the information available to the API to return, in bytes.

Bytes returned. The actual length of information returned to the caller of the API.

CCSID of password. The CCSID of the password. The possible values are:

0	Use the default CCSID for the current process.
1 - 65533	Valid range of CCSID values.

CCSID of server job The CCSID of the job on the server system. The possible values are:

0	Use the default CCSID for the current process.
1 - 65533	Valid range of CCSID values.

Commit scope. The commitment definition scope. The possible values are:

**JOB*	The job-level commitment definition is started for the job.
**ACTGRP*	An activation-group-level commitment definition is started for the activation group associated with the program issuing the command. This value is allowed for connection type L only. To simulate activation group commit scope in a remote environment, multiple remote connections must be used.
» **XA*	The XA-level commitment definition is started for the job. This value is not allowed with connection type L and/or commitment control N. «

Commitment control. The commit level to be used. The possible values are:

C	*CHG: Every record read for update (for a file opened under commitment control) is locked. If a record is changed, added, or deleted, that record remains locked until the transaction is committed or rolled back. Records that are accessed for update operations, but are released without being changed, are unlocked.
S	*CS: Every record accessed for files opened under commitment control is locked. A record that is read, but not changed or deleted, is unlocked when a different record is read. Records that are changed, added, or deleted are locked until the transaction is committed or rolled back.
A	*ALL: Every record accessed for files opened under commitment control is locked until the transaction is committed or rolled back.
N	*NONE: Commitment control should not be started.

Connection type. The communications type to use for the connection. The possible values are:

L	Local connection: Only one local connection per job may be open at a time. If a second local connection is attempted in the job, the connection actually will be made over UNIX domain sockets.
O	OptiConnect
T	TCP/IP sockets
U	UNIX domain sockets

Connection type used. The connection type that was actually used for the connection.

Connection handle. A unique handle number for the connection. The maximum number of connections per job that may be open at one time is 30.

Convert endian data. Whether integer data should be converted from OS/400 big-endian format to Windows PC little-endian format. The field is only used by the Client Access Express version of this API when returning data into the SQL Descriptor Area (SQLDA). The possible values are:

0	Do not convert endian data. If the API is called from an OS/400 application, you must code '0' for this field.
1	Convert endian integer data from big-endian to little-endian.

Job-associated user data. Data to associate with the server job that allows the job to be found using the QxdaFindJob API.

Job-suspension user data. Data associated with the current job to allow the job to be suspended independent of an entire system suspension.

Length of job-associated user data. The length of the job-associated data passed.

Length of job-suspension user data. The length of the job-suspension user data passed. This parameter must be set to 0 if the allow job suspension parameter is N.

Length of password associated with user profile. The length of the user profile password passed. The maximum length for the password is 512 bytes. Passwords can have a maximum of 128 characters. 512 bytes can accommodate 128 double bytes characters with a shift-in, shift-out pairing.

Length of user profile data. The length of the user profile data passed.

» Lock timeout value. Timeout value for locks in seconds. «

Offset to job-associated user data. The offset from the beginning of the input structure to the job-associated user data in the input structure, in bytes.

Offset to job-suspension user data. The offset from the beginning of the input structure to the job-suspension user data in the input structure, in bytes. This value must be 0 if the allow job suspension parameter is set to N.

Offset to password associated with user profile. The offset from the beginning of the input structure to the password associated with the user profile in the input structure, in bytes.

Offset to user profile data. The offset from the beginning of the input structure to the user profile data in the input structure, in bytes.

Password associated with user profile data. The password to be used in conjunction with the user profile to connect to the server system.

Relational database (RDB) name The relational database on the server system to which the connection should be made. This field should be blank padded to 18 characters or unpredictable results may occur. If the field is set to all blanks, the connection will be made to the *LOCAL (SYSBAS) RDB on the server system. If the server system does not have any active independent ASPs, an error will be signaled for any RDB that is not defined as the the *LOCAL RDB.

Relational database (RDB) specified Specifies whether the relational database (RDB) name field was provided. Possible values are:

0	The relational database (RDB) name field was not specified.
1	The relational database (RDB) name was specified.

Reserved. Reserved field; it must be initialized to 0x00.

Server job name. The job name of the database server job. If this is a local connection, this will be the current job name.

Server job number. The job number of the database server job. If this is a local connection, this will be the current job number.

Server job user name. The user name of the database server job's initial user. If this is a local connection, this will be the current job's initial user.

Server system name. The name of the system to which to connect. For connection type O, this is the current system name as displayed on the Display Network Attributes (DSPNETA) display on the server system. For connection type T, this is the server system name as displayed in the TCP/IP host table. It must be initialized to blanks for all other connection types. If the server system name is the local system, the connection actually will be made locally.

» **SQL hexadecimal constants.** This corresponds to the SET OPTION SQLCURRULE option and controls how hexadecimal constants are treated within SQL statements.

0	Corresponds to a SQL CURRULE value of *DB2 where hexadecimal constants are treated as character data.
1	Corresponds to a SQL SQLCURRULE value of *STD where hexadecimal constants are treated as binary data.

«

SQLDA cache size. The number of SQL descriptor areas to store for later reuse. An improvement in performance will be seen if an SQL descriptor area can be reused.

» Transaction manager information. Transaction manager name. «

User profile data. The name of the user profile to use to connect to the server system.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	Error Message Text
CPF180C E	Function &1 not allowed.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C21 E	Format name &1 is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFAE14 E	Cannot allocate &1 bytes.
CPFB751 E	Parameter &1 passed not correct.
CPFB752 E	Internal error in &1 API.
CPFB753 E	Required OptiConnect support not installed.
CPFB754 E	Unable to open connection. » Reason code &1.«
CPFB757 E	The connection is suspended.
CPFB758 E	The EDRS server system has been switched.

API introduced: V4R4

Top | "Database and File APIs," on page 1 | APIs by category

Disconnect from EDRS Server (QxdaDisconnectEDRS) API

Required Parameter Group:

1	Connection handle
Input	Binary(4)
2	Additional disconnection options
Input	Binary(4)
3	Error code
I/O	Char(*)

Service Program: QXDAEDRS

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 289

The Disconnect from EDRS Server (QxdaDisconnectEDRS) API is used to end a connection to a server system.

Authorities and Locks

None.

Required Parameter Group

Connection handle

INPUT; BINARY(4)

The handle number of the connection to end. The connection handle must have been generated by the QxdaConnectEDRS API in the current job and activation group.

Additional disconnection options

INPUT; BINARY(4)

The following are valid disconnection options:

0 - QXDA_DISCONNECT_COMMIT	Commit all uncommitted database work when the connection is ended.
1 - QXDA_DISCONNECT_ROLLBACK	Roll back all uncommitted database work when the connection is ended.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	**Error Message Text**
CPF180C E	Function &1 not allowed.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB750 E	Connection handle specified not valid.
CPFB751 E	Parameter &1 passed not correct.
CPFB752 E	Internal error in &1 API.
CPFB756 E	Rollback operation performed.
CPFB757 E	The connection has been suspended.

API introduced: V4R4

Top | "Database and File APIs," on page 1 | APIs by category

Find EDRS Job (QxdaFindEDRSJob) API

Required Parameter Group:

1	Connection handle
Input	Binary(4)
2	Job-associated user data
Input	Char(*)
3	Length of job-associated user data
Input	Binary(4)
4	Receiver variable
Output	Char(*)
5	Length of receiver variable
Input	Binary(4)
6	Receiver variable format
Input	Char(8)
7	Number of jobs found
Output	Binary(4)
8	Number of jobs returned
Output	Binary(4)
9	Error code
I/O	Char(*)

Service Program: QXDAEDRS

Default Public Authority: *USE

Threadsafe: Conditional; see “Usage Notes” on page 292

The Find EDRS Job (QxdaFindEDRSJob) API is used to find all jobs with user-defined data associated with the Connect to EDRS Server (QxdaConnectEDRS) API that matches the data passed to this API.

Authorities and Locks

None.

Required Parameter Group

Connection handle
INPUT; BINARY(4)

The handle number of the connection in which to find jobs. The connection handle must have been generated by the Connect to EDRS Server (QxdaConnectEDRS) API in the current job and activation group.

Job-associated user data
INPUT; CHAR(*)

User data that also was passed to the Connect to EDRS Server (QxdaConnectEDRS) API. This may be the complete user data or only a part of it. If it is only part, it must be the beginning of the user data string.

Length of job-associated user data
INPUT; BINARY(4)

The length of the user data to compare.

Receiver variable
OUTPUT; CHAR(*)

Space for the job information to be returned. This information is returned as an array of QJBI0100 structures, one for each job found. For the format of each array element, see "QJBI0100 Format."

Length of receiver variable
INPUT; BINARY(4)

Length (in bytes) of the receiver variable provided to return information about the jobs found.

Receiver variable format
INPUT; CHAR(8)

The format of the structure in which to return information about the jobs found. The possible value is:

QJBI0100	Basic receiver variable structure.

Number of jobs found
OUTPUT; BINARY(4)

The number of jobs found with associated user data that matches the user data passed in. This is the total number found, even if the information for all the jobs cannot fit in the space provided.

Number of jobs returned
OUTPUT; BINARY(4)

The actual number of jobs for which information was returned.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

QJBI0100 Format

The following table shows the structure of the QJBI0100 format. For more details about the fields in this table, see "Field Descriptions" on page 292.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Process ID
4	4	CHAR(10)	Job name
14	E	CHAR(10)	Job user name
24	18	CHAR(6)	Job number
30	1E	CHAR(16)	Internal job identifier
46	2E	CHAR(2)	Reserved

Field Descriptions

Internal job identifier. The internal job identifier. This value is sent to other APIs to speed the process of locating the job on the system.

Job name. The name of the job found.

Job number. The number of the job found.

Job user name. The name of the initial user of the job found.

Process ID. The process ID (PID) of the job found.

Reserved. Reserved field; it must be initialized to 0x00.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	Error Message Text
CPF180C E	Function &1 not allowed.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB750 E	Connection handle specified not valid.
CPFB751 E	Parameter &1 passed not correct.
CPFB752 E	Internal error in &1 API.
CPFB756 E	Rollback operation performed.
CPFB757 E	The connection is suspended.
CPFB758 E	The EDRS server system has been switched.

API introduced: V4R4

Top | "Database and File APIs," on page 1 | APIs by category

Forget XA (QxdaXAForget) API

Required Parameter Group:

1	Connection handle
Input	Binary(4)
2	Transaction branch identifier
Input	Char(*)
3	Return value
Output	Binary(4)
4	Flags
Input	Binary(4)
5	Error code
I/O	Char(*)

Service Program: QXDAEDRS

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 294

A transaction manager calls QxdaXAForget() to forget about a heuristically-completed transaction branch. After this call, the Transaction Branch Identifier is no longer valid. The connection does not have to be associated with the transaction in any way.

Authorities and Locks

None.

Required Parameter Group

Connection handle
INPUT; BINARY(4)

The handle number of the connection on which to perform the XA operation. The connection handle must have been generated by the QxdaConnectEDRS API in the current job and activation group.

Transaction branch identifier

The transaction branch identifier. This identifier is generated by the transaction manager when it starts the transaction branch. The max length is 140 bytes.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Format identifier
4	4	BINARY(4)	Length of global transaction identifier
8	8	BINARY(4)	Length of branch qualifier
12	C	CHAR(128)	Transaction branch identifier

Format identifier:	Identifies the naming format being used. If OSI CCR naming is used, the format identifier should be set to 0. If some other format is used, a the value should be greater than 0. A value of -1 means the transaction branch identifier is null.
Length of global transaction identifier:	Identifies the first of at most two contiguous components comprised by the data field. This specifies the number of bytes (1-64) starting at the first byte of the data element. This is not null-terminated.
Length of branch qualifier:	Identifies the second of at most two contiguous components comprised by the data field. This specifies the number of bytes (1-64) starting at the first byte after the global transaction identifier. This is not null-terminated.
Transaction branch identifier:	Contains the data of the transaction branch identifier.

Return value
OUTPUT; BINARY(4)

The return value of the XA operation.

-7	[XAER_RMFAIL]	An error occurred that makes the resource manager unavailable.
-6	[XAER_PROTO]	**xa_forget()** was not successful. Function was called in an improper context.
-5	[XAER_INVAL]	**xa_forget()** was not successful. Incorrect arguments were specified.
-4	[XAER_NOTA]	The specified xid is not known by the resource manager.
-3	[XAER_RMERR]	**xa_forget()** was not successful. The resource manager detected an error when forgetting the transaction branch.
-2	[XAER_ASYNC]	**xa_forget()** was not successful. The resource manager does not support asynchronous operations.
0	[XA_OK]	**xa_forget()** was successful.

Flags INPUT; BINARY(4)

Indicator of how to perform the XA operation. The following are valid settings of flags:

TMNOFLAGS:	x00000000	Perform the forget operation normally.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB760 E	Error encountered in &2 API, return code &1.

Related Information

The QxdaXA APIs follow the NTS (transactions with transaction scoped locks) model. For more information regarding the NTS model, see XA APIs.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include qxdaedrs.h

main()
{
err_t     err;
Qxda_CDBI0100_t *istr = (Qxda_CDBI0100_t *)instruct;
Qxda_CDBO0100_t  ostr;
int ostrsize;
XID inxid;
char gtrid[8] = "TestXA";
char bqual[6] = "Test";
inxid.formatID = 0;
inxid.gtrid_length = 6;
inxid.bqual_length = 4;
int create = XDA_SQL_TXN_CREATE;
int timeoutval = 60;
Qsq_sqlca_t      myca;
int   stmt1len = 21;
int flags = 0;
int ret_val;
_CPYBYTES(dtaara, gtrid, inxid.gtrid_length);
_CPYBYTES(dtaara + inxid.gtrid_length, bqual, inxid.bqual_length);
_CPYBYTES(inxid.data, dtaara, inxid.gtrid_length + inxid.bqual_length);
XID *xid = &inxid;




 memset(istr, '\0', sizeof(Qxda_CDBI0100_t));


 istr->Connection_Type = 'U';
 memset(istr->Server_Name, ' ', 256);
 istr->Commitment_Control = 'S';
 _CPYBYTES(istr->Commit_Scope, "*XA                                ", 10);
 istr->Allow_Suspend = 'Y';
 memset(istr->RDB_Specified, '0', 1);
 istr->SQLDA_Cache_Size = 10;
 istr->Offset_Job_Data = sizeof(Qxda_CDBI0100_t);
 istr->Length_Job_Data = 7;
 _CPYBYTES((char *)istr + istr->Offset_Job_Data, "CONNECT", 7);
 istr->Offset_Suspend_Data = istr->Offset_Job_Data + istr->Length_Job_Data;
 istr->Length_Suspend_Data = 7;
 memset(istr->RDB_Name, ' ', 18);
 _CPYBYTES(istr->TM_Info, "TM_Name         ", 10);
 istr->LockTimeoutVal = 10;
 ostrsize = sizeof(Qxda_CDBO0100_t);
 QxdaConnectEDRS(istr, "CDBI0100", &ostr, &ostrsize,
                  "CDBO0100", &err);


 QxdaSetXaConnection(&ostr.Connection_Handle, xid, &ret_val, &create, &timeoutval, &err);

QxdaProcessImmediateEDRS(&ostr.Connection_Handle,
              "SELECT * FROM MYTABLE", &stmt1len, &myca, &err);

 QxdaXAForget(&ostr.Connection_Handle, xid, &ret_val, &flags, &err);

}
```

API introduced: V5R3

Top | "Database and File APIs," on page 1 | APIs by category

Prepare XA (QxdaXAPrepare) API

Required Parameter Group:

1	Connection handle
Input	Binary(4)
2	Transaction branch identifier
Input	Char(*)
3	Return value
Output	Binary(4)
4	Flags
Input	Binary(4)
5	Error code
I/O	Char(*)

Service Program: QXDAEDRS

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 298

A transaction manager calls QxdaXAPrepare() to request that a resource manager prepare for commitment any work performed on behalf of the transaction branch identifier. The resource manager places all resources used in the transaction branch in a state that the changes can be made permanently when it later receives the QxdaXACommit() request. The connection does not have to be associated with the transaction in any way.

Authorities and Locks

None.

Required Parameter Group

Connection handle
INPUT; BINARY(4)

The handle number of the connection on which to perform the XA operation. The connection handle must have been generated by the QxdaConnectEDRS API in the current job and activation group.

Transaction branch identifier
INPUT; CHAR(*)

The transaction branch identifier. This identifier is generated by the transaction manager when it starts the transaction branch. The max length is 140 bytes.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Format identifier
4	4	BINARY(4)	Length of global transaction identifier
8	8	BINARY(4)	Length of branch qualifier
12	C	CHAR(128)	Transaction branch identifier

Format identifier:	Identifies the naming format being used. If OSI CCR naming is used, the format identifier should be set to 0. If some other format is used, a the value should be greater than 0. A value of -1 means the transaction branch identifier is null.
Length of global transaction identifier:	Identifies the first of at most two contiguous components comprised by the data field. This specifies the number of bytes (1-64) starting at the first byte of the data element. This is not null-terminated.
Length of branch qualifier:	Identifies the second of at most two contiguous components comprised by the data field. This specifies the number of bytes (1-64) starting at the first byte after the global transaction identifier. This is not null-terminated.
Transaction branch identifier:	Contains the data of the transaction branch identifier.

Return value
OUTPUT; BINARY(4)

The return value of the XA operation. The following return codes indicate that the resource manager has rolled back the work done on this transaction branch.

100	[XA_RBROLLBACK]	The transaction branch was rolled back for an unspecified reason.
101	[XA_RBCOMMFAIL]	The transaction was rolled back because a communications failure occurred within the resource manager.
102	[XA_RBDEADLOCK]	The transaction was rolled back because a deadlock condition was detected within the resource manager.
103	[XA_RBINTEGRITY]	The transaction was rolled back because the resource manager detected a violation of the integrity of its resources.
104	[XA_RBOTHER]	The resource manager rolled back the transaction branch for a reason not on this list.

105	[XA_RBPROTO]	The transaction was rolled back because a protocol error occurred in the resource manager.
106	[XA_RBTIMEOUT]	The transaction was rolled back because a timeout occurred in the resource manager.
107	[XA_RBTRANSIENT]	The transaction was rolled back because a transient error was detected in the resource manager.

All other return codes:

-7	[XAER_RMFAIL]	An error occurred that makes the resource manager unavailable.
-6	[XAER_PROTO]	**xa_prepare()** was not successful. Function was called in an improper context.
-5	[XAER_INVAL]	**xa_prepare()** was not successful. Incorrect arguments were specified.
-4	[XAER_NOTA]	The specified xid is not known by the resource manager.
-3	[XAER_RMERR]	**xa_prepare()** was not successful. The resource manager detected an error when preparing the transaction branch.
-2	[XAER_ASYNC]	**xa_prepare()** was not successful. The resource manager does not support asynchronous operations.
0	[XA_OK]	**xa_prepare()** was successful.
3	[XA_RDONLY]	The transaction branch was read-only and has been committed.

Flags INPUT; BINARY(4)

Indicator of how to perform the XA operation. The following are valid settings of flags:

TMNOFLAGS:	x00000000	Perform the prepare operation normally.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	**Error Message Text**
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB760 E	Error encountered in &2 API, return code &1.

Related Information

The QxdaXA APIs follow the NTS (transactions with transaction scoped locks) model. For more information regarding the NTS model, see XA APIs.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include qxdaedrs.h

main()
{
err_t     err;
Qxda_CDBI0100_t *istr = (Qxda_CDBI0100_t *) instruct;
Qxda_CDBO0100_t  ostr;
int ostrsize;
XID inxid;
char gtrid[8] = "TestXA";
char bqual[6] = "Test";
inxid.formatID = 0;
inxid.gtrid_length = 6;
inxid.bqual_length = 4;
int create = XDA_SQL_TXN_CREATE;
int end = XDA_SQL_TXN_END;
int timeoutval = 60;
Qsq_sqlca_t      myca;
int   stmt1len = 21;
int flags = 0;
int ret_val;
_CPYBYTES(dtaara, gtrid, inxid.gtrid_length);
_CPYBYTES(dtaara + inxid.gtrid_length, bqual, inxid.bqual_length);
_CPYBYTES(inxid.data, dtaara, inxid.gtrid_length + inxid.bqual_length);
XID *xid = &inxid;




 memset(istr, '\0', sizeof(Qxda_CDBI0100_t));


 istr->Connection_Type = 'U';
 memset(istr->Server_Name, ' ', 256);
 istr->Commitment_Control = 'S';
 _CPYBYTES(istr->Commit_Scope, "*XA                      ", 10);
 istr->Allow_Suspend = 'Y';
 memset(istr->RDB_Specified, '0', 1);
 istr->SQLDA_Cache_Size = 10;
 istr->Offset_Job_Data = sizeof(Qxda_CDBI0100_t);
 istr->Length_Job_Data = 7;
 _CPYBYTES((char *)istr + istr->Offset_Job_Data, "CONNECT", 7);
 istr->Offset_Suspend_Data = istr->Offset_Job_Data + istr->Length_Job_Data;
 istr->Length_Suspend_Data = 7;
 memset(istr->RDB_Name, ' ', 18);
 _CPYBYTES(istr->TM_Info, "TM_Name         ", 10);
 istr->LockTimeoutVal = 10;
 ostrsize = sizeof(Qxda_CDBO0100_t);


 QxdaConnectEDRS(istr, "CDBI0100", &ostr, &ostrsize,
                 "CDBO0100", &err);


 QxdaSetXaConnection(&ostr.Connection_Handle, xid, &ret_val, &create, &timeoutval, &err);

QxdaProcessImmediateEDRS(&ostr.Connection_Handle,
              "SELECT * FROM MYTABLE", &stmt1len, &myca, &err);
 QxdaSetXaConnection(&ostr.Connection_Handle, xid, &ret_val, &end, &timeoutval, &err);


 QxdaXAPrepare(&ostr.Connection_Handle, xid, &ret_val, &flags, &err);

}
```

API introduced: V5R3

Top | "Database and File APIs," on page 1 | APIs by category

Process Command (QxdaProcessCommandEDRS) API

Required Parameter Group:

1	Connection handle	**Input**	Binary(4)
2	Command	**Input**	Char(*)
3	Length of command	**Input**	Binary(4)
4	Error code	**I/O**	Char(*)

Service Program Name: QXDAEDRS

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 301

The Process Command (QxdaProcessCommandEDRS) API is used to run a system command on the database server system. The command is called exactly as passed, without coded character set identifier (CCSID) conversion.

Authorities and Locks

Any command
: *USE

Library of the command
: *EXECUTE

Required Parameter Group

Connection handle
INPUT; BINARY(4)

The handle number of the connection on which to call the command. The connection handle must have been generated by the QxdaConnectEDRS API in the current job and activation group.

Command
INPUT; CHAR(*)

The command you want to run entered as a character string. If the command contains blanks, it must be enclosed in apostrophes. The maximum length of the string is 32702 characters.

Length of command
INPUT; BINARY(4)

The length of the command to run.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	Error Message Text
CPF180C E	Function &1 not allowed.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFAE14 E	Cannot allocate &1 bytes.
CPFB750 E	Connection handle specified not valid.
CPFB752 E	Internal error in &1 API.
CPFB756 E	Rollback operation performed.
CPFB757 E	The connection is suspended.
CPFB758 E	The EDRS server system has been switched.
xxxnnnn E	Any escape message issued by any command may be returned.

API introduced: V4R4

Top | "Database and File APIs," on page 1 | APIs by category

Process Immediate SQL Statement (QxdaProcessImmediateEDRS) API

Required Parameter Group:

1	Connection handle	
Input	Binary(4)	
2	SQL statement	
Input	Char(*)	
3	Length of SQL statement	
Input	Binary(4)	
4	SQL communications area	
Output	Char(136)	
5	Error code	
I/O	Char(*)	

Service Program: QXDAEDRS

Default Public Authority: *USE

nbsp;Threadsafe: Conditional; see "Usage Notes" on page 303

The Process Immediate SQL Statement (QxdaProcessImmediateEDRS) API is used to run an SQL statement on the database server. The statement is processed exactly as provided, without coded character set identifier (CCSID) conversion.

Authorities and Locks

None.

Required Parameter Group

Connection handle
INPUT; BINARY(4)

The handle number of the connection on which to process the SQL statement. The connection handle must have been generated by the QxdaConnectEDRS API in the current job and activation group.

SQL statement
INPUT; CHAR(*)

The SQL statement to process.

Length of SQL statement
INPUT; BINARY(4)

The length of the SQL statement passed.

SQL communications area
OUTPUT; CHAR(136)

Returns diagnostic information. It includes the SQLCODE variable, indicating whether an error has occurred. If SQLCODE has a value of 0 after a call to this API, the function was successful.

The format of this structure is standard and is described more completely in the DB2 UDB for iSeries SQL Programming Concepts topic and the DB2 UDB for iSeries SQL Reference topic.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	Error Message Text
CPF180C E	Function &1 not allowed.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB750 E	Connection handle specified not valid.
CPFB752 E	Internal error in &1 API.
CPFB756 E	Rollback operation performed.
CPFB757 E	The connection is suspended.
CPFB758 E	The EDRS server system has been switched.

API introduced: V4R4

Top | “Database and File APIs,” on page 1 | APIs by category

Process Remote Extended Dynamic SQL (QxdaProcessExtDynEDRS) API

Required Parameter Group:

1	Connection handle	**Input**	Binary(4)
2	SQL descriptor area	**Input**	Char(*)
3	SQL communications area	**Output**	Char(136)
4	QSQPRCED function template format	**Input**	Char(8)
5	QSQPRCED function template	**Input**	Char(*)
6	Receiver variable	**Output**	Char(*)
7	Length of receiver variable	**Input**	Binary(4)
8	Receiver variable format	**Input**	Char(8)
9	Additional options	**Input**	Binary(4)
10	Error code	**I/O**	Char(*)

Service Program Name: QXDAEDRS

Default Public Authority: *USE

Threadsafe: Conditional; see “Usage Notes” on page 306

The Process Remote Extended Dynamic SQL (QxdaProcessExtDynEDRS) API is used to perform extended dynamic SQL operations on the database server system. The SQL operations are performed by the Process Extended Dynamic SQL (QSQPRCED) API.

Authorities and Locks

See the “Process Extended Dynamic SQL (QSQPRCED) API” on page 476 documentation for authorities required.

Required Parameter Group

Connection handle
INPUT; BINARY(4)

The handle number of the connection on which to perform the extended dynamic SQL operation. The connection handle must have been generated by the QxdaConnectEDRS API in the current job and activation group.

SQL descriptor area
INPUT; CHAR(*)

Passes information about the variables being used on a specific SQL statement. The SQLDA is used for passing the address, data type, length, and CCSID for variables on an OPEN, EXECUTE, FETCH, or DESCRIBE function.

The format of the structure is standard and is described more completely in the DB2 UDB for iSeries SQL Programming Concepts and DB2 UDB for iSeries SQL Reference topics.

SQL communications area
OUTPUT; CHAR(136)

Returns diagnostic information. It includes the SQLCODE variable, indicating whether an error has occurred. If SQLCODE has a value of 0 after a call to this API, the function was successful.

The format of this structure is standard and is described more completely in the DB2 UDB for iSeries SQL Programming Concepts and DB2 UDB for iSeries SQL Reference books.

QSQPRCED function template format
INPUT; CHAR(8)

The format of the function template being used. » The formats of the QSQPRCED function templates are standard. For possible values and complete descriptions, see the "Process Extended Dynamic SQL (QSQPRCED) API" on page 476 topic. « The SQLP0100, SQLP0110, SQLP0200, SQLP0210, and SQLP0500 QSQPRCED formats are not supported by the QxdaProcessExtDynEDRS API.

QSQPRCED function template
INPUT; CHAR(*)

Determines the function to perform, the requested statement to process, and the SQL package to be used. It also contains the text of the statement, which is required for the PREPARE function. For the format of this parameter, see the "Process Extended Dynamic SQL (QSQPRCED) API" on page 476 documentation.

Receiver variable
OUTPUT; CHAR(*)

The structure in which to return information about the connection. For the format of this parameter, see "EXDO0100 Format" on page 306.

Length of receiver variable
INPUT; BINARY(4)

The number of bytes that the calling program provides for the receiver variable data.

Receiver variable format
INPUT; CHAR(8)

The format of the receiver variable template being used. The possible value is:

EXDO0100	Basic receiver variable

Additional options
INPUT; BINARY(4)

The following are valid options. The binary OR operation can be used to use more than one of these options together.

0x00000000 - 0 - QXDA_EXTDYN_NOOPTS

0x00000001 - 1 - QXDA_CREATE_OBJECTS
: When preparing a statement into an SQL package, create the library and SQL package if they do not already exist. This option is valid only for QSQPRCED functions 2 and 9; it is ignored for all other functions. When this option is specified, all parameters required by the Process Extended Dynamic SQL (QSQPRCED) API for function 1 must be provided in the QSQPRCED function template.

0x00000010 - 16 - QXDA_FIND_STMT
: If this option is specified and the statement name parameter of the QSQPRCED function template is blank, a search is performed to see whether a prepared statement already exists in the specified library and package with the same statement text as the current text. If not, a unique statement name is generated and returned in the Statement name field of the receiver variable.

0x00000100 - 256 - QXDA_DEFER_OPEN
: Defer the open operation until a fetch operation is performed, when possible. The system determines whether the open operation can be deferred. This option is valid only for QSQPRCED function 4, and only when a remote connection type is used. This option is ignored in all other cases. It will cause a failure at fetch time if the fetch operation that immediately follows an open operation using this option is not from the same cursor as the open operation.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

EXDO0100 Format

The following table shows the information returned in the EXDO0100 format. For more details about the fields in this table, see "Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(18)	Statement name

Field Descriptions

Bytes available. The length of the information available to the API to return, in bytes.

Bytes returned. The actual number of bytes returned.

Statement name. The statement name generated when the QXDA_FIND_STMT option is specified.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	Error Message Text
CPF180C E	Function &1 not allowed.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C21 E	Format name &1 is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFAE14 E	Cannot allocate &1 bytes.
CPFB750 E	Connection handle specified not valid.
CPFB752 E	Internal error in &1 API.
CPFB756 E	Rollback operation performed.
CPFB757 E	The connection is suspended.
CPFB758 E	The EDRS server system has been switched.
CPFB759 E	Cursor not valid for operation.

API introduced: V4R4

Top | "Database and File APIs," on page 1 | APIs by category

Recover XA (QxdaXARecover) API

Required Parameter Group:

1	Connection handle
Input	Binary(4)
2	Array of transaction branch identifiers
Output	Array of Char(140)
3	Array entries allocated
Input	Binary(4)
4	Return value
Output	Binary(4)
5	Flags
Input	Binary(4)
6	Error code
I/O	Void(*)

Service Program: QXDAEDRS

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 309

A transaction manager calls QxdaXARecover() during recovery to obtain a list of transaction branch identifiers that are currently in a prepared or heuristically completed state. Multiple calls to this function can be made in a single recovery scan. The flags parameter defines when a recovery scan should start or end. The connection does not have to be associated with the transaction in any way.

Authorities and Locks

None.

Required Parameter Group

Connection handle
INPUT; BINARY(4)

The handle number of the connection on which to perform the XA operation. The connection handle must have been generated by the QxdaConnectEDRS API in the current job and activation group.

Array of transaction branch identifiers
OUTPUT; CHAR(*)

An array into which the resource manager places XIDs, each containing a max size of 140 bytes, for transaction branches in prepared or heuristically completed states.

Array entries allocated
INPUT; BINARY(4)

The number of transaction branch identifiers that can be returned into the Array of transaction branch identifiers.

Return value
OUTPUT; BINARY(4)

The return value of the XA operation. Possible return values:

-6	[XAER_PROTO]	**xa_recover()** was not successful. Function was called in an improper context.
-5	[XAER_INVAL]	**xa_recover()** was not successful. Incorrect arguments were specified.
-3	[XAER_RMERR]	**xa_recover()** was not successful. The resource manager detected an error when retrieving the transaction branch.

>=0	The total number of XIDs returned in the xids array.

Flags
INPUT; BINARY(4)

Indicator of how to perform the XA operation. The following are valid settings of flags:

TMSTARTRSCAN:	x01000000	Start a recovery scan and position the cursor to the start of the list. Transaction Branch Identifiers are returned from that point.
TMENDRSCAN:	x00800000	End a recovery scan after returning the XIDs. If this flag is used with the *TMSTARTRSCAN* flag, then a single xa_recover() call starts and ends the recovery scan.
TMNOFLAGS:	x00000000	Continue a recovery scan. Transaction Branch Identifiers are returned starting at the current cursor position.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB760 E	Error encountered in &2 API, return code &1.

Related Information

The QxdaXA APIs follow the NTS (transactions with transaction scoped locks) model. For more information regarding the NTS model, see XA APIs.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include qxdaedrs.h

main()
{
err_t     err;
Qxda_CDBI0100_t *istr = (Qxda_CDBI0100_t *)instruct;
Qxda_CDBO0100_t  ostr;
int ostrsize;
XID xids[10];
Qsq_sqlca_t      myca;
int   stmt1len = 21;
int flags = TMSTARTRSCAN + TMENDRSCAN;
int count = 30;
int ret_val;




 memset(istr, '\0', sizeof(Qxda_CDBI0100_t));


 istr->Connection_Type = 'U';
 memset(istr->Server_Name, ' ', 256);
 istr->Commitment_Control = 'S';
 _CPYBYTES(istr->Commit_Scope, "*XA                     ", 10);
 istr->Allow_Suspend = 'Y';
 memset(istr->RDB_Specified, '0', 1);
 istr->SQLDA_Cache_Size = 10;
 istr->Offset_Job_Data = sizeof(Qxda_CDBI0100_t);
 istr->Length_Job_Data = 7;
 _CPYBYTES((char *)istr + istr->Offset_Job_Data, "CONNECT", 7);
 istr->Offset_Suspend_Data = istr->Offset_Job_Data + istr->Length_Job_Data;
 istr->Length_Suspend_Data = 7;
 memset(istr->RDB_Name, ' ', 18);
 _CPYBYTES(istr->TM_Info, "TM_Name         ", 10);
 istr->LockTimeoutVal = 10;
 ostrsize = sizeof(Qxda_CDBO0100_t);


 QxdaConnectEDRS(istr, "CDBI0100", &ostr, &ostrsize,
                  "CDBO0100", &err);


 QxdaXARecover(&ostr.Connection_Handle, xids, &count, &ret_val, &flags, &err);

}
```

《 API introduced: V5R3

Top | "Database and File APIs," on page 1 | APIs by category

Roll Back EDRS Server (QxdaRollbackEDRS) API

Required Parameter Group:

1	Connection handle
Input	Binary(4)
2	Additional rollback options
Input	Binary(4)
3	SQL communications area
Output	Char(136)
4	Error code
I/O	Char(*)

Service Program: QXDAEDRS

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 311

The Roll Back EDRS Server (QxdaRollbackEDRS) API is used to roll back transactions on the database server.

Authorities and Locks

None.

Required Parameter Group

Connection handle
INPUT; BINARY(4)

The handle number of the connection on which to perform the rollback operation. The connection handle must have been generated by the QxdaConnectEDRS API in the current job and activation group.

Additional rollback options

INPUT; BINARY(4)

The following are valid rollback options:

0	QXDA_ROLLBACK_WORK
1	QXDA_ROLLBACK_WITH_HOLD

SQL communications area

OUTPUT; CHAR(136)

Returns diagnostic information. It includes the SQLCODE variable, indicating whether an error has occurred. If SQLCODE has a value of 0 after a call to this API, the function was successful.

The format of this structure is standard and is described morecompletely in the DB2 UDB for iSeries SQL Programming Concepts and DB2 UDB for iSeries SQL Reference books.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	**Error Message Text**
CPF180C E	Function &1 not allowed.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB750 E	Connection handle specified not valid.
CPFB751 E	Parameter &1 passed not correct.
CPFB752 E	Internal error in &1 API.
CPFB757 E	The connection is suspended.
CPFB758 E	The EDRS server system has been switched.

API introduced: V4R4

Top | "Database and File APIs," on page 1 | APIs by category

Rollback XA (QxdaXARollback) API

Required Parameter Group:

1	Connection handle
Input	Binary(4)
2	Transaction branch identifier
Input	Char(*)
3	Return value
Output	Binary(4)
4	Flags
Input	Binary(4)
5	Error code
I/O	Char(*)

Service Program: QXDAEDRS

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 314

A transaction manager calls QxdaXARollback() to roll back work performed on behalf of the transaction branch identifier. A transaction branch is capable of being rolled back until it has been successfully committed. The connection does not have to be associated with the transaction in any way.

Authorities and Locks

None.

Required Parameter Group

Connection handle
INPUT; BINARY(4)

The handle number of the connection on which to perform the XA operation. The connection handle must have been generated by the QxdaConnectEDRS API in the current job and activation group.

Transaction branch identifier
INPUT; CHAR(*)

The transaction branch identifier. This identifier is generated by the transaction manager when it starts the transaction branch. The max length is 140 bytes.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Format identifier
4	4	BINARY(4)	Length of global transaction identifier
8	8	BINARY(4)	Length of branch qualifier
12	C	CHAR(128)	Transaction branch identifier

Format identifier:	Identifies the naming format being used. If OSI CCR naming is used, the format identifier should be set to 0. If some other format is used, a the value should be greater than 0. A value of -1 means the transaction branch identifier is null.
Length of global transaction identifier:	Identifies the first of at most two contiguous components comprised by the data field. This specifies the number of bytes (1-64) starting at the first byte of the data element. This is not null-terminated.
Length of branch qualifier:	Identifies the second of at most two contiguous components comprised by the data field. This specifies the number of bytes (1-64) starting at the first byte after the global transaction identifier. This is not null-terminated.
Transaction branch identifier:	Contains the data of the transaction branch identifier.

Return value
OUTPUT; BINARY(4)

The return value of the XA operation. The following return codes indicate that the resource manager rolled back the work done on this transaction branch. These values are typically returned when the transaction branch was previously marked rollback-only:

100	[XA_RBROLLBACK]	The transaction branch was rolled back for an unspecified reason.
101	[XA_RBCOMMFAIL]	The transaction was rolled back because a communications failure occurred within the resource manager.
102	[XA_RBDEADLOCK]	The transaction was rolled back a deadlock condition was detected within the resource manager.
103	[XA_RBINTEGRITY]	The transaction was rolled back the resource manager detected a violation of the integrity of its resources.
104	[XA_RBOTHER]	The resource manager rolled back the transaction branch for a reason not on this list.
105	[XA_RBPROTO]	The transaction was rolled back a protocol error occurred in the resource manager.
106	[XA_RBTIMEOUT]	The transaction was rolled back a timeout occurred in the resource manager.
107	[XA_RBTRANSIENT]	The transaction was rolled back a transient error was detected in the resource manager.

The following values may be returned for all flags settings.

-7	[XAER_RMFAIL]	An error occurred that makes the resource manager unavailable.
-6	[XAER_PROTO]	**xa_rollback()** was not successful. Function was called in an improper context.
-5	[XAER_INVAL]	**xa_rollback()** was not successful. Incorrect arguments were specified.
-4	[XAER_NOTA]	The specified xid is not known by the resource manager.
-3	[XAER_RMERR]	**xa_rollback()** was not successful. The resource manager detected an error when rolling back the transaction branch.

-2	[XAER_ASYNC]	**xa_rollback()** was not successful. The resource manager does not support asynchronous operations.
0	[XA_OK]	**xa_rollback()** was successful.
5	[XA_HEURMIX]	Work on the transaction branch was partially committed and partially rolled back.
6	[XA_HEURRB]	Work on the transaction branch was heuristically rolled back.
7	[XA_HEURCOM]	Work on the transaction branch was heuristically committed.
8	[XA_HEURHAZ]	Work on the transaction branch may have been heuristically completed.

Flags INPUT; BINARY(4)

Indicator of how to perform the XA operation. The following are valid settings of flags:

TMNOFLAGS:	x00000000	Perform the rollback operation normally.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB760 E	Error encountered in &2 API, return code &1.

Related Information

The QxdaXA APIs follow the NTS (transactions with transaction scoped locks) model. For more information regarding the NTS model, see XA APIs.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include qxdaedrs.h

main()
{
err_t     err;
Qxda_CDBI0100_t *istr = (Qxda_CDBI0100_t *)instruct;
Qxda_CDBO0100_t  ostr;
int ostrsize;
XID inxid;
char gtrid[8] = "TestXA";
char bqual[6] = "Test";
inxid.formatID = 0;
inxid.gtrid_length = 6;
inxid.bqual_length = 4;
int create = XDA_SQL_TXN_CREATE;
int end = XDA_SQL_TXN_END;
int timeoutval = 60;
Qsq_sqlca_t      myca;
int   stmt1len = 21;
int flags = 0;
int ret_val;
_CPYBYTES(dtaara, gtrid, inxid.gtrid_length);
_CPYBYTES(dtaara + inxid.gtrid_length, bqual, inxid.bqual_length);
_CPYBYTES(inxid.data, dtaara, inxid.gtrid_length + inxid.bqual_length);
XID *xid = &inxid;




 memset(istr, '\0', sizeof(Qxda_CDBI0100_t));


 istr->Connection_Type = 'U';
 memset(istr->Server_Name, ' ', 256);
 istr->Commitment_Control = 'S';
 _CPYBYTES(istr->Commit_Scope, "*XA                      ", 10);
 istr->Allow_Suspend = 'Y';
 memset(istr->RDB_Specified, '0', 1);
 istr->SQLDA_Cache_Size = 10;
 istr->Offset_Job_Data = sizeof(Qxda_CDBI0100_t);
 istr->Length_Job_Data = 7;
 _CPYBYTES((char *)istr + istr->Offset_Job_Data, "CONNECT", 7);
 istr->Offset_Suspend_Data = istr->Offset_Job_Data + istr->Length_Job_Data;
 istr->Length_Suspend_Data = 7;
 memset(istr->RDB_Name, ' ', 18);
 _CPYBYTES(istr->TM_Info, "TM_Name         ", 10);
 istr->LockTimeoutVal = 10;
 ostrsize = sizeof(Qxda_CDBO0100_t);


 QxdaConnectEDRS(istr, "CDBI0100", &ostr, &ostrsize,
                  "CDBO0100", &err);


 QxdaSetXaConnection(&ostr.Connection_Handle, xid, &ret_val, &create, &timeoutval, &err);

QxdaProcessImmediateEDRS(&ostr.Connection_Handle,
              "SELECT * FROM MYTABLE", &stmt1len, &myca, &err);

 QxdaSetXaConnection(&ostr.Connection_Handle, xid, &ret_val, &end, &timeoutval, &err);

 QxdaXARollback(&ostr.Connection_Handle, xid, &ret_val, &flags, &err);

}
```

API introduced: V5R3

Top | "Database and File APIs," on page 1 | APIs by category

Set Connection (QxdaSetConnection) API

Required Parameter Group:

1	Connection handle	
Input	Binary(4)	
2	Transaction branch identifier	
Input	Char(*)	
3	Return value	
Output	Binary(4)	
4	Operation	
Input	Binary(4)	
5	Timeout value	
Input	Binary(4)	
6	Error code	
I/O	Char(*)	

Service Program: QXDAEDRS

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 318

A transaction manager calls QxdaSetConnection() to perform XA-transaction related operations for a particular connection.

Authorities and Locks

None.

Required Parameter Group

Connection handle
INPUT; BINARY(4)

The handle number of the connection on which to perform the XA operation. The connection handle must have been generated by the QxdaConnectEDRS API in the current job and activation group.

Transaction branch identifier
INPUT; CHAR(*)

The transaction branch identifier. This identifier is generated by the transaction manager when it starts the transaction branch. The max length is 140 bytes.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Format identifier
4	4	BINARY(4)	Length of global transaction identifier
8	8	BINARY(4)	Length of branch qualifier

Offset		Type	Field
Dec	Hex		
12	C	CHAR(128)	Transaction branch identifier

Format identifier:	Identifies the naming format being used. If OSI CCR naming is used, the format identifier should be set to 0. If some other format is used, a the value should be greater than 0. A value of -1 means the transaction branch identifier is null.
Length of global transaction identifier:	Identifies the first of at most two contiguous components comprised by the data field. This specifies the number of bytes (1-64) starting at the first byte of the data element. This is not null-terminated.
Length of branch qualifier:	Identifies the second of at most two contiguous components comprised by the data field. This specifies the number of bytes (1-64) starting at the first byte after the global transaction identifier. This is not null-terminated.
Transaction branch identifier:	Contains the data of the transaction branch identifier

Return value
OUTPUT; BINARY(4)

The return value of the XA operation. Possible return values:

0	*SQL_SUCCESS*
-1	*SQL_ERROR*

Operation
INPUT; BINARY(4)

Valid operations are:

QXDA_SQL_TXN_FIND:	(1)	Allows a connection to attach to an existing transaction.
QXDA_SQL_TXN_CREATE:	(2)	Creates a new transaction.
QXDA_SQL_TXN_SUSPEND:	(3)	Disassociates the connection from the transaction and only closes the native files (end with suspend). The SQL cursors remain open so a connection can pick up the transaction and continue where it left off.
QXDA_SQL_TXN_END:	(4)	Disassociates the connection from the transaction (no suspend).

Note: Both *QXDA_SQL_TXN_FIND* and *QXDA_SQL_TXN_CREATE* are similar to a **xa_start**; whereas, *QXDA_SQL_TXN_SUSPEND* and *QXDA_SQL_TXN_END* are similar to a **xa_end**.

Timeout value
INPUT; BINARY(4)

Timeout value for the transaction in seconds. If the Timout Value is not greater than 0, then the default system timeout value will be used. This value must be >= 0 and applies to the *XDA_SQL_TXN_CREATE* operation.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

This function may be called from the initial thread of a job only.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB760 E	Error encountered in &2 API, return code &1.

Related Information

The QxdaXA APIs follow the NTS (transactions with transaction scoped locks) model. For more information regarding the NTS model, see XA APIs.

Example

See Code disclaimer information for information pertaining to code examples.

```
#include qxdaedrs.h

main()
{
err_t     err;
Qxda_CDBI0100_t *istr = (Qxda_CDBI0100_t *)instruct;
Qxda_CDBO0100_t  ostr;
int ostrsize;
XID inxid;
char gtrid[8] = "TestXA";
char bqual[6] = "Test";
inxid.formatID = 0;
inxid.gtrid_length = 6;
inxid.bqual_length = 4;
int create = XDA_SQL_TXN_CREATE;
int timeoutval = 60;
int ret_val;
_CPYBYTES(dtaara, gtrid, inxid.gtrid_length);
_CPYBYTES(dtaara + inxid.gtrid_length, bqual, inxid.bqual_length);
_CPYBYTES(inxid.data, dtaara, inxid.gtrid_length + inxid.bqual_length);
XID *xid = &inxid;





 memset(istr, '\0', sizeof(Qxda_CDBI0100_t));

 istr->Connection_Type = 'U';
 memset(istr->Server_Name, ' ', 256);
 istr->Commitment_Control = 'S';
 _CPYBYTES(istr->Commit_Scope, "*XA                      ", 10);
 istr->Allow_Suspend = 'Y';
 memset(istr->RDB_Specified, '0', 1);
 istr->SQLDA_Cache_Size = 10;
 istr->Offset_Job_Data = sizeof(Qxda_CDBI0100_t);
 istr->Length_Job_Data = 7;
 _CPYBYTES((char *)istr + istr->Offset_Job_Data, "CONNECT", 7);
 istr->Offset_Suspend_Data = istr->Offset_Job_Data + istr->Length_Job_Data;
 istr->Length_Suspend_Data = 7;
 memset(istr->RDB_Name, ' ', 18);
 _CPYBYTES(istr->TM_Info, "TM_Name         ", 10);
 istr->LockTimeoutVal = 10;
 ostrsize = sizeof(Qxda_CDBO0100_t);


 QxdaConnectEDRS(istr, "CDBI0100", &ostr, &ostrsize,
                 "CDBO0100", &err);


 QxdaSetConnection(&ostr.Connection_Handle,  xid, &ret_val, &create, &timeoutval, &err);
}
```

API introduced: V5R3

Top | “Database and File APIs,” on page 1 | APIs by category

Set XDA Options (QxdaSetOptions) API

Required Parameter Group:

1	Input structure	Input	Char(*)
2	Input structure format	Input	Char(8)
3	Error code	I/O	Char(*)

Service Program: QXDAEDRS

Default Public Authority: *USE

Threadsafe: Conditional; see “Usage Notes” on page 321

The Set XDA Options (QxdaSetOptions) API is used to set options related to XDA.

Authorities and Locks

None.

Required Parameter Group

Input structure
INPUT; CHAR(*)

The structure is used to set options related to XDA. For the format of this parameter, see “SETO0100 Format” on page 320.

Input structure format
INPUT; CHAR(8)

The format of the input structure template being used. The possible value is:

SETO0100	Set Options structure.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

SETO0100 Format

The following table shows the information to pass in the SETO0100 format. For more details about the fields in this table, see "Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(1)	Threadsafe
1	1	CHAR(1)	Trace mode
2	2	CHAR(2)	Reserved1
4	4	BINARY(4)	Trace file size
8	8	BINARY(4)	Offset to the trace configuration data
12	C	BINARY(4)	Length of the trace configuration data
16	10	CHAR(112)	Reserved2
		CHAR(*)	Trace configuration data

Field Descriptions

Length of the trace configuration data. The length of the trace configuration data passed.

Offset to the trace configuration data. The offset from the beginning of the input structure to the trace configuration data in the input structure, in bytes.

Reserved1. Reserved field; it must be initialized to 0x00.

Reserved2. Reserved field; it must be initialized to 0x00.

Threadsafe. Indicates whether thread safety should be used in XDA. This should be specified before QxdaConnectEDRS is called. The default is not to use thread safety in XDA.

0	Do not use thread safety in XDA.
1	Use thread safety in XDA.

Trace configuration data. Indicates whether the trace function should trace network flows, or trace network data blocks, or both. The normal trace function must be enabled in order to take advantage of the trace configurations. The following trace options can be specified together with a comma separator:

TRACENET	Trace network data flow in XDA.
TRACEDATABLOCK	Trace values inserted or removed from network data blocks.

Trace file size. Indicates the size of the trace file in megabytes. The file is wrapped after the size is reached. The default is not to wrap the trace file. The maximum size is 2048 MB (2 GB).

0	Do not wrap the trace file in XDA.

Trace mode. Indicates the mode of the trace function that should be used in XDA. There are not any specific instructions for use. The default is not to enable tracing functionality in XDA.

0	Do not enable tracing in XDA.
1	Enable tracing at an information level in XDA.
2	Enable tracing at an error level in XDA.
3	Enable tracing at the verbose level in XDA.

Usage Notes

This function may be called from the initial thread of a job only for thread safety. This function can be called from any thread to enable the trace function. The trace function will be enabled for all connections within a given job.

Error Messages

Message ID	**Error Message Text**
CPF3C21 E	Format name &1 is not valid.
CPFB751 E	Parameter &1 passed not correct.
CPFB752 E	Internal error in &1 API.

« API introduced: V5R3 with PTF

Top | “Database and File APIs,” on page 1 | APIs by category

Database Miscellaneous APIs

The Database Miscellaneous APIs are:

- “Bring Database Records (QDBBRCDS) API” on page 322 (QDBBRCDS) asynchronously brings database physical file records into main storage.
- “Change Cross Reference CCSID (QDBCXRC) API” on page 324 (QDBCXRC) changes the CCSID of the system cross reference files.
- “Create Database Hash (QCreateDatabaseHash) API” on page 325 (QCreateDatabaseHash) sets up the environment to enable the Run Database Hash (QDBRUNHA) API for a physical file that has a uniquely keyed logical file built over it.
- “Query (QQQQRY) API” on page 328 (QQQQRY) gets a set of database records that satisfy a database query request.
- “Run Database Hash (QDBRUNHA) API” on page 370 (QDBRUNHA) allows the user to FETCH, UPDATE, DELETE, and INSERT data into existing database files using an alternative access method.

«

Top | Database and File APIs | APIs by category

Bring Database Records (QDBBRCDS) API

Required Parameter Group:

1	Qualified database physical file name	
Input	Char(20)	
2	Database member name	
Input	Char(10)	
3	Relative record number array	
Input	Array of Binary(4)	
4	Number of records to bring	
Input	Binary(4)	

Optional Parameter Group 1:

	5
Error code	I/O
Char(*)	

Default Public Authority: *USE

Threadsafe: Yes

The Bring Database Records (QDBBRCDS) API asynchronously brings database physical file records into main storage. You can use the QDBBRCDS API only with database file type *PF. DDM files and logical files are not supported. If a distributed file is specified, only the records on the local system are brought. File overrides do not affect the specified file, library, or member names.

Authorities and Locks

Library Authority
*EXECUTE

File Authority
*OBJOPR

File Lock
None

Required Parameter Group

Qualified database physical file name
INPUT; CHAR(20)

The name of the database physical file containing the specified member whose information is to be retrieved and the library in which it is located. The first 10 characters contain the database physical file name; the second 10 characters contain the library name.

You can use these special values for the library name:

**CURLIB*	The job's current library
**LIBL*	The library list

Database physical file member name
INPUT; CHAR(10)

The name of the database physical file member for which information is to be retrieved.

Special values follow:

**FIRST*	The first database physical file member found.
**LAST*	The last database physical file member found.

Relative record array
INPUT; CHAR(*)

A array of unsigned Binary(4) variables that contain the relative record numbers that should be brought.

If an invalid relative record number is specified, it is tolerated and no error is returned. All relative record numbers prior to the invalid relative record number in the array are processed. All relative record numbers after the invalid relative record number in the array are not processed.

Number of records in the array
INPUT; BINARY(4)

An unsigned Binary(4) variable that contains the number of relative record numbers in the array. The number of relative record numbers must not exceed 1000.

Optional Parameter Group 1

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3CF1 E	Error code parameter not valid.
CPF3C23 E	Object &1 is not a database file.
CPF3C26 E	File &1 has no members.
CPF3C3A E	Value for parameter &2 for API &1 not valid.
CPF3C90 E	Literal value cannot be changed.
CPF8100 E	All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9800 E	All CPF98xx messages could be signaled. xx is from 01 to FF.

Top | "Database and File APIs," on page 1 | APIs by category

API Introduced: V3R7

Change Cross Reference CCSID (QDBCXRC) API

Required Parameter Group:

1	CCSID	
Input	Binary(4)	
2	Error Code	
I/O	Char(*)	

Default Public Authority: *EXCLUDE

Threadsafe: No

The Change Cross Reference CCSID (QDBCXRC) API changes the CCSID of the system cross reference files.

Note: To be able to use this API, the system must be in the restricted state.

Authorities and Locks

API Public Authority
*EXCLUDE

User Special Authority
*ALLOBJ

Required Parameter Group

CCSID
INPUT; BINARY(4)

The desired coded character set identifier (CCSID) for the system cross reference files. Valid values for the CCSID range from 1 through 65535. For a list of valid CCSID values, see the Globalization topic. Only CCSID values to which a job can be changed are accepted.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	**Error Message Text**
CPF24B4 E	Severe error while addressing parameter list.
CPF3202 E	File &1 in library &2 in use.
CPF328A E	System not in proper state to do requested operation.
CPF3CF1 E	Error code parameter not valid.
CPF9803 E	Cannot allocate object &1 in library &2.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V5R2

Top | “Database and File APIs,” on page 1 | APIs by category

Create Database Hash (QCreateDatabaseHash) API

Required Parameter Group:

1
Hash name

Input
Char(10)

2
Physical file

Input
Char(10)

3
Physical file library

Input
Char(10)

4
Logical file

Input
Char(10)

5
Logical file library

Input
Char(10)

6
Expression

Input
Char(64)

7
Number-of-keys

Input
Binary(4)

8
Key ranges

Input
Char(*)

Service Program Name: QDBCRTHA

Default Public Authority: *USE

Threadsafe: No

The Create Database Hash (QCreateDatabaseHash) API sets up the environment to enable the Run Database Hash (QDBRUNHA) API for a physical file that has a uniquely keyed logical file built over it. The logical file may have up to five integer keys associated with it. It is called as a function call of the form 'xx = Qcrtdbha(parameter-list)', where xx is a long integer and the parameter list is as defined here. The value of xx is set to a return code as defined in the "Returned Value" on page 327 topic.

Authorities and Locks

HASH User Space in Library QUSRSYS
: *OBJOPR, *READ, and *UPDATE

» **Library Authority**
: *EXECUTE

File Authority
: *OBJOPR

File Lock
: *SHRNUP «

Required Parameter Group

Hash name
INPUT; CHAR(10)

The hash name to be created. A unique name must be selected for each hash function that will be used on the system.

Physical file
INPUT; CHAR(10)

The name of the physical file that will be accessed using the hash.

Physical file library
INPUT; CHAR(10)

The name of the library where the physical file resides.

Logical file
INPUT; CHAR(10)

The name of the logical file that will be used to build the hash. The logical file must be uniquely keyed.

Logical file library
INPUT; CHAR(10)

The name of the library where the logical file resides.

Expression
INPUT; CHAR(64)

A valid mathematical expression that uses all the key values of a uniquely keyed logical file to determine the hash value for a particular record. The special value of *DFT can be used to allow the API to create an expression based on expected cardinalities (number of expected unique values for each key) of the keys in the logical file. Possible values are:

**DFT* The system default expression is used (requires the use of the number of keys parameter and the key ranges parameter).

expression
The user-defined expression is used. For example: (where K1, K2, ... K5 are the names of the key fields used in the logical file)

Number of keys
INPUT; BINARY(4)

The number of keys used in the logical file.

Key ranges
INPUT; CHAR(*)

A two-value structure with up to five occurrences, containing the names of the key fields followed by the expected cardinality of the key. For more details, see "Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
		CHAR(10)	Name of key
		BINARY(4)	Cardinality

Field Descriptions

Name of key. The name of the key field that is used in the logical file, which is referenced in this API. The names are for documentation purposes only.

Cardinality. The number of sequential values expected to be used for each key, respectively. The cardinality values are required if *DFT has been specified for the value of the expression parameter.

Returned Value

The returned value contains a numeric indication as to what took place during the request to add a hash function. The possible values are:

0	No errors.
-2	The physical file has multiple formats. The create database hash function cannot be completed.
-3	The logical file is not uniquely keyed. The create database hash function cannot be completed.
-4	The logical file does not correlate to the physical file specified. The create database hash function cannot be completed.
-5	The wrong number of keys was specified for the logical file. The create database hash function cannot be completed.
-99	Another error was encountered and ignored. See job log for details.

Error Messages

Only the error conditions listed in the "Returned Value" are monitored. No error messages other than the value of the return code parameter are returned.

API introduced: V4R3

Top | "Database and File APIs," on page 1 | APIs by category

Query (QQQQRY) API

Required Parameter Group:

1	Query option requested
Input	Char(10)
2	User file control block
I/O	Char(*)
3	Query definition template
I/O	Char(*)
4	Literal values
I/O	Char(*)
5	Access plan control block
I/O	Char(48)
6	Error code
I/O	Char(*)

Default Public Authority: *EXCLUDE

Threadsafe: Conditional; see “Usage Notes” on page 368.

The Query (QQQQRY) API gets a set of database records that satisfies a database query request. Using this API you can do all the things you could do with the Open Query File (OPNQRYF) command. You can also perform subqueries, perform unions, and use SQL host variables.

The QQQQRY API can be used to do any combination of the following database functions:

- Join records from more than one file, member, and record format. The join operation that is performed may be equal or nonequal in nature.
- Calculate new field values by using numeric and character operations on field values and constants.
- Group records by like values of one or more fields, and calculate aggregate functions, such as minimum field value and average field value, for each group.
- Select a subset of the available records. Selection can be done both before and after grouping the records.
- Arrange result records by the value of one or more key fields.

You can use this API to run a query, create an access plan, or get information from the query definition template (QDT). When you run the query, the API uses the information you provide with the query definition template to extract information and data from the database. Creating an access plan makes it possible to run the query with better performance. Checking the query definition template allows you to validate the values in this query definition template.

The format definition is part of the query definition template and can be created and saved with extracted information by the Retrieve Database File Description (QDBRTVFD) API. When you are finished using the QQQQRY API, you should close the file (using the Close File (CLOF) command) to free up resources.

Another part of the query definition template is the access plan for the query. Using this API with the Create Query Access Plan (CRTQAP) value of the query option requested parameter, you can build an access plan to more efficiently run a query more than once. You can then use the access plan control block parameter to point to the access plan. This greatly improves the time it takes to perform subsequent runs of this query using this API and the RUNQRY option. Every time a query is run, the system first checks to see if an access plan has been specified. If one has, that is what is used to get the data requested by the query. If no access plan has been specified, a new one is built dynamically.

Authorities and Locks

User Space Authority
: *CHANGE

Library Authority
: *EXECUTE

File Authority
: *OBJOPR

User Space Lock
: *SHRRD

Required Parameter Group

Query option requested
INPUT; CHAR(10)

One of three options to be used:

- RUNQRY
- Run query
- CRTQAP
- Create query access plan
- CHKQDT
- Check query definition templates

User file control block
I/O; CHAR(*)

One or more selected options for input and output of the specified query. This parameter need only be used along with the RUNQRY query option. See “User File Control Block (QDBUFCB_T) Structure” on page 362 for a list of available options.

Query definition template
I/O; CHAR(*)

The information required to create objects that are used to query a database. It contains feedback information from the creation of objects. If a pointer to the access plan is specified, the corresponding query definition templates must also be specified.

Literal values
I/O; CHAR(*)

This parameter is used to put into effect SQL host variables. When SQL host variables are used, this is a list of constant values used to run a query. If this parameter is to be ignored, a null pointer can be specified for the parameter. Once the literal value is specified on a call, it must always be specified.

Access plan control block
I/O; CHAR(48)

A string of bytes that point to the access plan control block and give the size the access plan requires. This parameter must be specified for the RUNQRY query option when you want to specify an access plan and for the CRTQAP query option. The format for this parameter is:

PTR(SPP)	A space pointer that indicates the area of storage that contains the access plan. This area must begin on a 16-byte boundary and be all zeros.
Bin(4)	The size of storage needed to contain the access plan.
Char(28)	Reserved.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Data Structures

The QQQQRY API uses information in four structures to carry out a query. All structures are used together to perform the function you have selected using the query option requested parameter. The names of these structures are:

QDBQH_T	Query definition template
Qdb_Qddfmt_t	Format definition template
QDBUFCB_T	User file control block
QQQVALS_T	Values for query variable fields

The following sections show you in a general way how this information is structured.

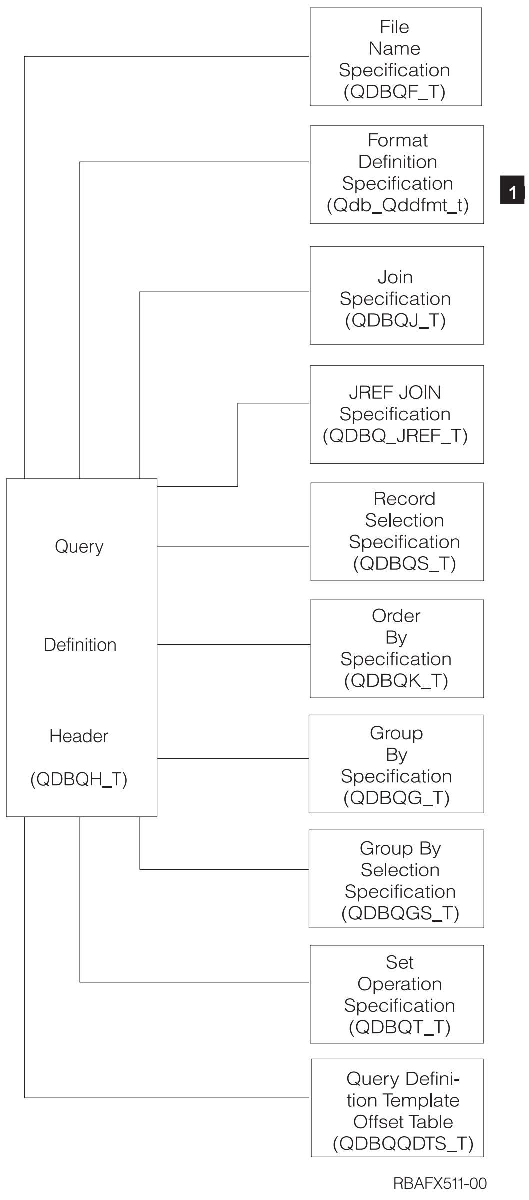
Query Definition Template (QDBQH_T)

The query definition template provides information about the query that is to be performed. QDBQH_T Format (page 330) shows the general layout of this format.

Notice the box marked with a **(1)** in QDBQH_T Format (page 330). The topic "Format Definition Template (Qdb_Qddfmt_t)" on page 361 provides the layout of the entire record format specification.

The offsets and descriptions of all the fields contained in this structure are shown in the following tables. You can see this source in member QQQQRY in the QSYSINC library.

QDBQH_T Format



File Name Specification (QDBQF_T)
Format Definition Specification (Qdb_Qddfmt_t)
1
Join Specification (QDBQJ_T)
JREF JOIN Specification (QDBQ_JREF_T)
Record Selection Specification (QDBQS_T)
Order By Specification (QDBQK_T)
Group By Specification (QDBQG_T)
Group By Selection Specification (QDBQGS_T)
Set Operation Specification (QDBQT_T)
Query Definition Template Offset Table (QDBQQDTS_T)
Query Definition Header (QDBQH_T)
RBAFX511-00

Query Definition Header (QDBQH_T)

This is the first structure and is located at offset zero. *(Ref #1.)*

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		BIN(4)	qdbqfilo	Offset to file, library, format, and member specifications.
4	4		BIN(4)	qdbqfldo	Offset to the record specifications. 0 indicates that the record specifications should be taken from the file format. 0 is not valid if there are multiple files in the file specification and this is not a group-by query.
8	8		BIN(4)	qdbqjoio	Offset to join specifications. 0 indicates that this is not a join query.
12	C		BIN(4)	qdbqselo	Offset to the record selection specifications. 0 indicates that no record selection is to occur.

Offset		Bit	Type	Variable Name	Field
Dec	**Hex**				
16	10		BIN(4)	qdbqkeyo	Offset to the order by specifications. 0 indicates that the records are returned in the file's access path order. If this is a join, the access path order of the first file in the file array is used. -1 indicates that the result records are returned in no guaranteed order; running the same query twice may produce a different result order.
20	14		BIN(4)	qdbqgrpo	Offset to the group-by-selection specifications. 0 indicates that no group by is to occur.
24	18		CHAR(4)	qdbqdt_1	Reserved.
28	1C		BIN(4)	qdbqgpso	Offset to the group by selection specifications. 0 indicates that no group by selection is to occur. If field qdbqgrpo is 0, this offset is ignored.
32	20		CHAR(1)	qdbqfin	Query completion indicator. *A* Query need not be completed before returning. The database attempts to minimize the entire query and retrieval time. Selection may be done at I/O time. *F* Query need not be complete before returning. The database attempts to minimize the time to get the first buffer of results. Selection may be done at I/O time. *M* Query need not be complete before returning; however, selection at I/0 time should be minimized so that long waits for the next selected records are minimized. *C* Query must be completed before returning. If this is a join, the records must be put in a temporary file.
33	21		CHAR(1)	qdbqtem	Query temporary result indicator. *N* Temporary results should be prohibited. *O* Temporary results are allowed but should be used only if necessary to do the query. If a read previous operation can be requested, then 0 must be used. *T* Temporary results are allowed but should be used only if necessary to do the query. However, if temporary results are used, then use the last TSORT method, which reads directly from its sort. This option cannot be specified if a read previous operation is to be used. *A* Temporary results are allowed and should be used if better performance can be achieved by using a temporary result. Use A when the user does not request previous records to be read.
34	22		CHAR(2)	qdbqattr	Query attributes.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
34	22	0	BIT(1)	qdbqnst	Status messages. Status messages are never sent for batch jobs. *0* Do not send status messages. *1* Send status messages during query and I/O processing.
34	22	1	BIT(1)	qdbqdist	Distinct records. *0* Do not produce distinct records. *1* Eliminate duplicate records from the query result.
34	22	2	BIT(1)	qdbqpgmd	Program-defined files. *0* Ignore interactive data definition utility (IDDU) data definitions for program-described files. *1* Use IDDU data definitions for program-described files.
34	22	3	BIT(1)	qdbqterr	Tolerate decimal data errors. *0* Decimal data errors result in an exception being issued. *1* Decimal data errors are ignored.
34	22	4	BIT(1)	qdbqdt_2	Reserved.
34	22	5	BIT(1)	qdbqintd	Integer division. *0* Do not perform integer division. *1* Perform integer division. Division of two integer (binary) numbers produces a zero precision result.
34	22	6	BIT(1)	qdbqdt_3	Reserved.
34	22	7	BIT(1)	qdbqchgx	Changed files exception. *0* No exception requested. *1* Send an exception when a queried file has changed since creation of the input access plan.
35	23	0	BIT(1)	qdbqsaap	Precision reduction (SAA$^{(R)}$) *0* Allow precision reduction. *1* Disallow reduction of the precision of a derived result. Instead, reduce significant digits when necessary.
35	23	1	BIT(1)	qdbqddmx	Distributed data management (DDM) files exception. *0* No exception requested. *1* Send an exception when a queried file is a DDM file.
35	23	2	BIT(1)	qdbqraut	Resolve authority. *0* Normal authority checking. User must have corresponding data authority for each open option. *1* Check for at least one data authority (read, add, update, or delete) regardless of the open options.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
35	23	3	BIT(1)	qdbqsqlb	SQL definition of binary. *0* Binary fields have digits as known by the database. *1* Binary fields in SQL tables and views have 11 digits if the binary is large and 5 digits if it is small.
35	23	4	BIT(1)	qdbqaltc	Alternate computation. *0* Do not use alternate computation. *1* Use alternate computation. Some derivations do not overflow as fast when no precision reduction (SAA) is allowed (qdbqsaap=1). Also, use the user-defined result field size for one-operation derivations (+, -, *, /).
35	23	5	BIT(1)	qdbqsubq	The query definition template contains at least one subquery. This does not span across unions.
35	23	6	BIT(1)	qdbqsubx	Subcharacters exception. This field specifies what to do if, during CCSID compatibility processing, a conversion occurs on the data such that information may be lost or misinterpreted. *0* Allow the query to finish. Information messages are returned if this condition occurs. *1* Send an exception. For literals and host variable values, the exception is sent during the open operation; check the query definition template or create an access plan of the query if subcharacters were used during the conversion of the value. For fields and conversion tables, the error occurs during I/O operations on the query if subcharacters are used.
35	23	7	BIT(1)	qdbqdt_4	Reserved.
36	24		BIN(2)	qdbqkunum	The NODUPKEY number of key fields. The database does not return any records with duplicate keys and determines this by using this number of key fields as a comparison length. -1 indicates that all the key fields are used as a comparison length. This field is not applicable if field qdbqkeyo is -1.
38	26		BIN(4)	qdbqnumrcd	This field is no longer supported. Its value will be ignored.
42	2A		BIN(4)	qdbqnxqo	Offset to next query definition template. This value is 0 if this is the only query definition template or if this is the last query definition template.
46	2E		BIN(4)	qdbqtoso	Offset to the set operation specifications. The set operation specifications can only be specified on the last query definition template. 0 indicates that no set operation is to occur. If only one query definition query template is specified, this offset is ignored.
50	32		CHAR(4)	qdbqdt_5	Reserved.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
54	36		BIN(4)	qdbqqdto	Offset to the query definition template table containing offsets to all query definition templates between unions.
58	3A		CHAR(8)	qdbqdt_6	Reserved. *(Ref #2.)*
66	42		CHAR(1)	qdbqdfmt	Date format used as the preferred format for validity checking a date string or when mapping a character field to a date. *X'FE'* Job default format *X'01'* USA format *X'03'* ISO format *X'05'* EUR format *X'07'* JIS format *X'17'* MDY format *X'18'* DMY format *X'19'* YMD format *X'1A'* JUL format When the value of this field is X'FE', the preferred format is obtained from the job attributes, which have the value X'17', X'18', X'19', or X'1A'. *(Ref #3.)*
67	43		CHAR(1)	qdbqdsep	Date separator used as the preferred format for validity checking a date string or when mapping a character field to a date. It is only set when field qdbqdfmt is X'FE', X'17', X'18', X'19', or X'1A'. *X'00'* Job default separator *X'EE'* Implied separator / Slash separator - Dash separator . Period separator , Comma separator *Blank* Blank separator When the value of this field is X'00', the preferred separator is obtained from the job attributes, which are one of the previously defined values except X'00' and X'EE'. When the value of this field is X'EE', the implied separator for the format is used. *(Ref #4.)*

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
68	44		CHAR(1)	qdbqtfmt	The time format used as the preferred format when validity checking a time string or when mapping a character field to a time. *X'FE'* Job default format *X'01'* USA format *X'03'* ISO format *X'05'* EUR format *X'07'* JIS format *X'1B'* HMS format When the value of this field is X'FE', the preferred format is obtained from the job attributes, which will have the value X'1B'. *(Ref #5.)*
69	45		CHAR(1)	qdbqtsep	The time separator used as the preferred separator when validity checking a time string or when mapping a character field to a time. It is only set when field qdbqtfmt is X'FE' or X'1B'. *X'00'* Job default separator *X'EE'* Implied separator *.* Period separator *,* Comma separator *Blank* Blank separator *:* Colon separator When the value of this field is X'00', the preferred separator is obtained from the job attributes, which are one of the values previously defined except X'00' and X'EE'. When the value of this field is X'EE', the implied separator for the format is used.
70	46		BIN(2)	qdbqcsdc	The coded character set identifier (CCSID) constant tag. If nonzero, this specifies the CCSID with which the literals in the query definition template should be tagged. If this field specifies a single-byte character set (SBCS) CCSID and a literal has double-byte character set (DBCS) data, the associated mixed CCSID of the SBCS CCSID is used for the literal. Conversely, if the literal is SBCS and this field specifies a mixed or DBCS CCSID, the associated SBCS CCSID is used for the literal.
72	48		CHAR(2)	qdbqdt_7	More flags.
72	48	0	BIT(1)	qdbqvlit	Variable length literal. *0* No *1* Yes
72	48	1	BIT(5)	qdbqdt_8	Reserved. *(Ref #6.)*

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
72	48	6	BIT(1)	qdbqopta	Optimize all indexes over the query files. *0* The optimizer determines how many indexes to consider when optimizing the query. The optimizer *times out* if the time spent optimizing becomes significant when compared to the time it takes for the query to run. *1* Optimize all indexes built over the query files. This may increase the time it takes for the optimization of the query to occur.
72	48	7	BIT(1)	qdbqmapbd	Reserved.
73	49	0	BIT(7)	qdbqdt_9	Reserved.
73	49	7	BIT(1)	qdbq_force_temp	Force query records to a temporary result. Note that this is only honored if set on for the last, outermost (that is, non-subquery) QDT of a union. It is ignored for all other QDTs of the query. *0* Normal processing. *1* Force results of entire query into a temporary result.
74	4A		CHAR(2)	QDBQDT_10	Reserved.
76	4C			qdbqfbk	Query feedback. The following information is returned on successful completion of the query.
76	4C		CHAR(2)	qdbqqtyp_t	Query status indicators.
76	4C	0	BIT(1)	qdbqtemp	Temporary result. *0* No temporary result. *1* Temporary result created.
76	4C	1	BIT(1)	qdbqcomp	Selection completion. *0* Selection is not complete. *1* Selection is complete. If selection is complete, the open feedback area contains the number of selected records. If selection is not complete, record selection may be performed while reading the records and the open feedback may indicate more records than are ultimately selected.
76	4C	2	BIT(1)	qdbqdt_11	Reserved.
76	4C	3	BIT(1)	qdbqacpi	Access plan indicator. *0* No access plan present. *1* Query definition template is or is part of an access plan.
76	4C	4	BIT(1)	qdbqsreg	This field is set on if the query definition template contains special registers CURRENT USER, CURRENT SERVER, or CURRENT TIMEZONE.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
76	4C	5	BIT(1)	qdbqsubw	Subcharacters warning. This field specifies whether during CCSID compatibility processing, subcharacters are used in the query. *0* No subcharacters are used. *1* Subcharacters are used in the query.
76	4C	6	BIT(1)	qdbqsqlt	SQL tables. *0* Not all SQL tables. *1* All SQL tables.
76	4C	7	BIT(1)	qdbqlblst	The library list was used to determine the referenced table. This is possible due to the use of the Override with Database File (OVRDBF) command.
77	4D	0	BIT(1)	qdbqcurdt	The query definition template contains CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP.
77	4D	1	BIT(7)	qdbqdt_12	Reserved.
78	4E		CHAR(28)	qdbqdt_13	Length of query definition structure.
78	4E		BIN(4)	qdbqdtln	Length of query definition.
82	52		CHAR(24)	qdbqdt_14	Reserved.
106	6A		CHAR(1)	qdbqdofmt	Date format for output date fields. *X'FE'* Job default format *X'FF'* Format specified with based-on field. *X'01'* USA format *X'03'* ISO format *X'05'* EUR format *X'07'* JIS format *X'17'* MDY format *X'18'* DMY format *X'19'* YMD format *X'1A'* JUL format If the data type Qddfftyp (page 88) is unknown (X'FFFF') and this field is X'FF', the format and separator are taken from those specified with the based-on field. If the data type Qddfftyp is date (X'000B'), the format and separator are taken from the extension of record formats Qddfdttf (page 91) and Qddfdtts (page 92). However, if Qddfdttf is X'FF', the format and separator are taken from qdbqdofmt and qdbqdosep. If either of these fields are not valid, it is an error. When the value of qdbqdofmt is X'FE', the format is obtained from the job attributes, which will have the value X'17', X'18', X'19', or X'1A'. Qddfftyp, Qddfdttf, and Qddfdtts are part of the QDBRTVFD include.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
107	6B		CHAR(1)	qdbqdosep	Date separator used as the output separator for fields. It is only set when qdbqdofmt is X'FE', X'17', X'18', X'19', or X'1A'. *X'00'* Job default separator *X'EE'* Implied separator / Slash separator - Dash separator . Period separator , Comma separator *Blank* Blank separator When the value of this field is X'00', the separator is obtained from the job attributes (any of the preceding values except X'00' and X'EE'). When the value of this field is X'EE', the implied separator for the format is used.
108	6C		CHAR(1)	qdbqtofmt	Time format for output time fields. *X'FE'* Job default format *X'FF'* Format specified with based-on field. *X'01'* USA format *X'03'* ISO format *X'05'* EUR format *X'07'* JIS format *X'1B'* HMS format If the data type Qddfftyp (page 88) is unknown (X'FFFF') and this field is X'FF', the format and separator are taken from those specified with the based-on field. If the data type Qddfftyp is time (X'000C'), the format and separator are taken from the extension of record formats Qddfdttf (page 91) and Qddfdtts (page 92). However, if Qddfdttf is X'FF', the format and separator are taken from qdbqtofmt and qdbqtosep. If either of these fields are not valid, it is an error. When the value of qdbqtofmt is X'FE', the format is obtained from the job attributes, which have the value X'1B'. Qddfftyp, Qddfdttf, and Qddfdtts are part of the QDBRTVFD include.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
109	6D		CHAR(1)	qdbqtosep	Time separator used as the output separator for fields. It is only set when qdbqtofmt is X'FE' or X'1B'. *X'00'* Job default separator *X'EE'* Implied separator *.* Period separator *,* Comma separator *:* Colon separator When the value of this field is X'00', the separator is obtained from the job attributes (any of the above values except X'00' and X'EE'). When the value of this field is X'EE', the implied separator for the format is used.
110	6E		CHAR(1)	qdbqtsofmt	Timestamp format for output timestamp fields. *X'FE'* Job default format *X'FF'* Format specified with based-on field *X'09'* SAA timestamp format If the data type Qddfftyp (page 88) is unknown (X'FFFF') and this field is X'FF', the format and separator are taken from those specified with the based-on field. If the data type Qddfftyp is timestamp (X'000D'), the format and separator are taken from the extension of record formats Qddfdttf (page 91) and Qddfdtts (page 92). However, if Qddfdttf is X'FF', the format and separator are taken from qdbqtsofmt and qdbqtsosep. If qdbqtsofmt contains a format that is not valid, it is an error. Qddfftyp, Qddfdttf, and Qddfdtts are part of the QDBRTVFD include.
111	6F		CHAR(1)	qdbqdt_15	Reserved.
112	70		BIN(4)	qdbq_optmrows	Optimization option. This field tells the optimizer that the user does not intend to retrieve more than *n* records from the query result. *n* can be any integer as long as it fits in a BIN(4) type. If the optimizer optimizes for *n* records, this could improve performance. Specifying a number does not mean the user cannot retrieve more than *n* records. It just tells the optimizer to optimize for only *n* records. For more information about the OPTIMIZE clause, see the DB2 UDB for iSeries SQL Reference topic.
116	74		CHAR(12)	qdbqdt_16	Reserved.
128	80		BIN(4)	qdbq_jrefo	Offset to JREF Join specification.
132	84		CHAR(44)	qdbqdt_65	Reserved.
176	B0		CHAR(2)	`qdbq_posnopts _t`	The ORed byte containing an indicator for every scrolling option that is used for this query. Scrolling option of next is always assumed.
176	B0	0	BIT(1)	`qdbq_posnopts _prior`	Scrolling to previous record is used.
176	B0	1	BIT(1)	`qdbq_posnopts _first`	Scrolling to first record is used.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
176	B0	2	BIT(1)	qdbq_posnopts _last	Scrolling to last record is used.
176	B0	3	BIT(1)	qdbq_posnopts _before	Scrolling to before the first record is used.
176	B0	4	BIT(1)	qdbq_posnopts _after	Scrolling to after the last record is used.
176	B0	5	BIT(1)	qdbq_posnopts _current	Retrieval of the current record is used.
176	B0	6	BIT(1)	qdbq_posnopts _relative	Scrolling to a record relative to the current record is used.
176	B0	7	BIT(9)	qdbqdt_17	Reserved.
178	B2		CHAR(1)	qdbq_ext_bits	Miscellaneous bits in the query definition header.
178	B2	0	BIT(1)	qdbq_ctlblk	An indicator that the caller will be control record blocking; therefore, ignore SEQONLY() overrides.
178	B2	1	BIT(1)	qdbq_norolb	Rollback HOLD can leave the position of this cursor as unknown.
178	B2	2	BIT(1)	qdbq_stream _cursor	The user of this query attempts to read records from this query as fast as possible.
178	B2	3	BIT(4)	qdbqdt_54	Reserved.
178	B2	7	BIT(1)	qdbqdt_18	Reserved.
179	B3		CHAR(1)	qdbq_ext_bits2	Miscellaneous bits in the query definition header.
179	B3	0	BIT(2)	qdbqdt_57	Reserved.
179	B3	2	BIT(1)	qdbq_trust _posn	Trust scrolling option *0* Query optimizer assumes that any type of cursor positioning may be done. *1* Settings of qdbq_posnopts can be trusted. The user that built this QDT has knowingly set the options and could experience problems if cursor positioning not indicated is attempted. This bit should be set on with qdbq_posnopts set to '0000'X to give the query optimizer more flexibility in choosing the best data access method and also to enable symmetric multiprocessing (SMP) methods such as parallel table scan and hash join.
179	B3	3	BIT(5)	qdbqdt_58	Reserved.
180	B4		CHAR(1)	qdbq_ext_bits3	Miscellaneous bits in the query definition header.
180	B4	0	BIT(1)	qdbqdt_62	Reserved.
180	B4	1	BIT(1)	qdbq_searched_ update	Indicator if this is a searched UPDATE QDT.
180	B4	2	BIT(1)	qdbq_searched_ delete	Indicator if this is a searched DELETE QDT.
180	B4	3	BIT(2)	qdbqdt_63	Reserved.
180	B4	5	BIT(1)	qdbq_drvtbl	Indicates that this QDT is part of a derived table QDT.
180	B4	6	BIT(2)	qdbqdt_64	Reserved.
181	B5		CHAR(15)	qdbqdt_19	Reserved.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
196	C4		CHAR(10)	qdbpopnid	Optional OPENID to identify this query. *FILE indicates the name of first or only file specified in the file specification. You can also specify a name to associate with opened query file.
206	CE		BIN(4)	qdbspcsize	API users specify size for space containing all API query definition templates. *(Ref #7.)*
210	D2		BIN(4)	qdbqnlss	Displacement to QQQNLSS_T structure ("Sequence, Tables, Names, and Parameters (QQQNLSS_T)") used for sort sequence information. This is an offset from the beginning of the query definition template. 0 indicates no QQQNLSS_T structure was passed. For nonviews, if this is a union or subquery, this field is ignored unless it is the first query definition template (for unions), or the outermost query definition template (for subqueries).
214	D6		BIN(2)	qdbqsrts	Sort sequence indicator. Possible values for this field follow. When you use value 2 or 3, you must include the QQQNLSS_T structure ("Sequence, Tables, Names, and Parameters (QQQNLSS_T)") at offset qdbqnlss. *0* *HEX *2* The table or CCSID passed in structure QQQNLSS_T. *3* The sort sequence and language identifier passed in structure QQQNLSS_T.
216	D8		CHAR(5)	qdbqdt_53	Reserved.
221	DD		CHAR(1)	qdbqic	Whether query allows index creation. *N* Index creation is not allowed. *Y or X'00'* Index creation is allowed.
222	DE		CHAR(178)	qdbqdt_20	Reserved.

Sequence, Tables, Names, and Parameters (QQQNLSS_T)

Sequence, tables, names, and parameters structure. The displacement to this structure from the beginning of structure QDBQH_T is an entry in the table at variable qdbqnlss (page 342).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(42)	qqqstb1	Sequence table name, library name, and language identifier.
0	0		CHAR(10)	qqqresv2	Reserved.
10	A		CHAR(20)	qqsrtseq	Sort sequence.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
10	A		CHAR(10)	qqqstbnm	Table name. Possible special values are: **JOB* The sort sequence of the job. **LANGIDUNQ* The unique-weight sort sequence table that is associated with the language identifier requested parameter. **LANGIDSHR* The shared-weight sort sequence table that is associated with the language identifier requested parameter. **HEX* The sort sequence according to the hexadecimal value of the characters.
20	14		CHAR(10)	qqqstbnl	Library name. Possible special values are: **LIBL* The library list. **CURLIB* The job's current library.
30	1E		CHAR(10)	qqqlangid	Language identifier. Possible values are: **JOB* The language identifier of the job. *xxx* 3-character language identifier. See Language identifiers and associated default CCSIDs for a complete list of language identifiers supported. *Blank* If blank, field qqqsrtseq cannot be *JOB, *LANGIDUNQ, or *LANGIDSHR.
40	28		CHAR(2)	qqqresv3	Reserved.
42	2A		CHAR(38)	qqqstb2	Reserved.
42	2A		CHAR(2)	qqqresv4	Reserved.
44	2C		CHAR(10)	qqqtbnm	Reserved.
54	36		CHAR(10)	qqqlbnm	Reserved.
64	40		CHAR(14)	qqqresv5	Reserved.
78	4E		BIN(2)	qqqtbl_ccsid	Sequence table CCSID. This field is only used when either qqqtbl is specified or qqqstboff is set for a DBCS sort sequence table.
80	50		CHAR(10)	qqqstbe1	User-specified DBCS sort sequence information.
80	50		CHAR(1)	qqqstbtyp	Type of DBCS sort sequence table. *X'00'* UCS-2 sort sequence table
81	51		CHAR(1)	qqqstbloc	Location of DBCS sort sequence table. *X'00'* Table is stored at qqqtbl. *X'01'* Table is stored at the DBCS sort sequence table offset (qqqstboff).
82	52		BIN(4)	qqqstblen	Length of DBCS sort sequence table. If an SBCS sort sequence table is specified, qqqstblen must be zero.
86	56		BIN(4)	qqqstboff	Offset to the DBCS sort sequence table from qqqstb1.
90	5A		CHAR(22)	qqqresv1	Reserved.
112	70		CHAR(256)	qqqtbl	User-specified sort sequence table.

File Name Specification (QDBQF_T)

File name specification. This structure defines the files, member, and formats that are used in the query. This structure is required.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0			qdbqfhdr	File specification header.
0	0		BIN(2)	qdbqfilnum	The number of files, libraries, formats, and members. *(Ref #8.)*
2	2		CHAR(1)	qdbqmfop	Multiple file option. This field is only applicable if the qdbqfilnum field is greater than one. *J* Inner join. No default values are supplied if a join value does not exist and no record is returned. *C* Partial outer join (file chaining). Default values are supplied if a join value does not exist. *E* Exception join. Default values are supplied if a join value does not exist. Only records with default values are returned.
3	3		CHAR(1)	qdbqmfor	Multiple file order option. This field is only applicable if field qdbqfilnum is greater than one, field qdbqmfop equals J, and there is no file-distinct processing. For each file specified in the file specifications, qdbqfdst equals 1. Partial-outer join, exception join, and file-distinct processing implies no join reordering. *A* Join the files in any order. The result order may vary even when rerunning the same query. *N* Join the files in the order they are specified.
4	4		CHAR(1)	qdbqdt_21	Flags.
4	4		BIT(1)	qdbqmfio	Multiple file I/O options allowed through this query. *0* Only allow read operations against the first file in the array (always read-only for secondary files). *1* Allow insert, update, or delete operations against the first file.
4	4	1	BIT(1)	qdbqmfjn	Join clause exists. An SQL JOIN clause syntax exists in this query.
4	4	2	BIT(6)	qdbqdt_22	Reserved.
5	5		CHAR(11)	qdbqdt_23	Reserved.
16	10		ARRAY(32) OF CHAR(64)	qdbqn	File, library, member, and format array. This structure is defined at QDBQN_T on “File, Library, Member, and Format Array (QDBQN_T).”

File, Library, Member, and Format Array (QDBQN_T)

File, library, member and format names array. This structure defines the file, library, member, and format names that are used in the query. This structure is required.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(40)	qdbqflmf	File, library, member, and record format names.
0	0		CHAR(10)	qdbqfile	File name. If an override is in effect to another file, the actual file name is returned in this field.
10	A		CHAR(10)	qdbqlib	Library name. If the special value *LIBL is used, the actual library name is resolved and returned in this field.
20	14		CHAR(10)	qdbqmbr	Member name. If the special values *FIRST or *LAST are used, the actual member name is resolved and returned in this field.
30	1E		CHAR(10)	qdbqfmt	Format name. If the special value *ONLY is used, the actual format name is resolved and returned in this field.
40	28		CHAR(1)	qdbqfflg	File specification flags.
40	28	0	BIT(1)	qdbqfdst	File-distinct flag. This field specifies, for the records that make up the join secondaries for a join query, whether only the first record or all records that satisfy the join conditions should participate in the join. This flag only applies to join secondary files (files 2 through *n*, where n equals the number of files in the join). *0* All records participate. *1* Only the first record participates.
40	28	1	BIT(1)	qdbqfujn	Unique join fanout. This field specifies whether the number of join records found can exceed 1. This field only applies to join secondary files (files 2 through *n*, where n equals the number of files in the join). *0* Multiple join records are allowed. *1* Only one join-to record may be found for this file.
40	28	2	BIT(1)	qdbqfgna	Reserved.
40	28	3	BIT(1)	qdbqfngn	Reserved.
40	28	4	BIT(1)	qdbqnmch	Name change indicator *0* The library, file, or member name in the specified query definition template (at offset qdbqfilo in structure qdbqh_t, "Query Definition Header (QDBQH_T)" on page 331) did not change as a result of an override. *1* The library, file, or member name in the specified query definition template (at field qdbqfilo (page 331)) changed due to an override.
40	28	5	BIT(1)	qdbqflbo	Library name overridden. *0* The library name in the specified query definition template (at offset qdbqfilo in structure qdbqh_t, "Query Definition Header (QDBQH_T)" on page 331) did not change as a result of an override. *1* The library name in the specified query definition template (at offset qdbqfilo (page 331)) changed to *LIBL due to an override, and the file was found using *LIBL as the library name. This bit is a feedback bit. The user of the query definition template should not set it.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
40	28	6	BIT(1)	qdbqf_nldft	Null or default. The type of values to be returned for unmatched records of a partial outer or exception join. *0* Return default values *1* Return NULL values
40	28	7	BIT(1)	qdbqdt_24	Reserved.
41	29		CHAR(1)	qdbqmfvw	Reserved.
42	2A		CHAR(1)	qdbqmfvw_spc	Reserved.
43	2B		BIN(2)	qdbqf_qdtnum	Index into the array of subquery offsets (QDBQQDT_T) for the derived table QDT.
45	2D		CHAR(19)	QDBQDT_25	Reserved.

Record Format Specification (QDBQR_T)

Record format specification. This structure defines the fields that are used in the query. The structure Qdb_Qddfmt_t is mapped by member QDBRTVFD in the QSYSINC library. If join is specified, this specification is required.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
			CHAR(*)	QDBQR	Record specifications.

Join Specification (QDBQJ_T)

Join specification. This structure defines how the files are joined by the query. One join specification exists for the entire query definition. A join specification entry consists of a from-field, a join operator, and a to-field. The join specification entries can be inserted in any order with respect to the file specifications.

If this is an **inner join** (the qdbqmfop field (page 344) equals J, and no join specifications are given for a particular to-file, the system searches the record selection specifications for any possible implied join specifications. If no join specifications can be derived from the record selection specifications, Cartesian product is used to do the join.

All join specifications can be given in the record selection specifications. In this case, it is not necessary to provide a join specification.

If this is a **partial-outer** or **exception join** (qdbqmfop equals C or E) and no join specifications are given for a particular to-file, the system uses Cartesian product to do the join. In addition, only one join operator can be specified for a particular to-file.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0			qdbqjhdr	Join specifications header.
0	0		BIN(4)	qdbqjln	Length of this join specification.
4	4		BIN(2)	qdbqjknum	Number of from-join and to-join field specifications.
6	6		CHAR(10)	qdbqdt_26	Reserved.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
16	10		ARRAY(1) OF CHAR(96)	qdbqjfld	Join specification array. Array of fields that define the from and to fields to use when joining. The structure is defined at QDBQJFLD_T on "Join Specification Array (QDBQJFLD_T)."
112	70		CHAR(*)	QDBQJNXT	Join field pair arrays. Displacement to join specifications array from structure QDBQJ_T (see structure QDBQJFLD_T on "Join Specification Array (QDBQJFLD_T)").

Join Specification Array (QDBQJFLD_T)

Join specification array. This structure is an array of fields that define the from and to fields to use when joining.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(30)	qdbqjfnm	Join from field name.
30	1E		BIN(2)	qdbqjfnum	Field join reference number. 0 indicates that the QDBQR_T format (see "Record Format Specification (QDBQR_T)" on page 346) is searched for the external field name. If the field is not found, the formats of the files in the file specification are searched. A value in this field indicates that the external field name is to be found in the file format referenced by using this value as an index into the file name specification structure, qdbqf_t, (see "File Name Specification (QDBQF_T)" on page 344). In any case, the field found must exist in a file joined prior to this file.
32	20		CHAR(2)	qdbqdt_27	Reserved.
34	22		CHAR(2)	qdbqjop	Join option. *EQ* Equal *GT* Greater than *LT* Less than *NE* Not equal *GE* Greater than or equal *LE* Less than or equal
36	24		CHAR(30)	qdbqjtnm	Join to field name. Note that only character and any DBCS fields may be joined to character and any DBCS fields, and only numeric fields may be joined to numeric fields. The lengths of the two fields need not be the same. However, if they are different, a warning is sent to the user indicating that padding occurred.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
66	42		BIN(2)	qdbqjtnum	Field join reference number. 0 indicates that the QDBQR_T format (see "Record Format Specification (QDBQR_T)" on page 346) is searched for the external field name. If the field is found, it must have been completely derived from the file associated with this join specification. If the field is not found, the format of the file associated with this join specification is searched. A value in this field indicates that the external field name is to be found in the file format referenced by using this value as an index into the file list. This value must reference the file associated with this join specification.
68	44		CHAR(1)	qdbqjpfmt	Reserved.
69	45		CHAR(1)	qdbqjpsep	Reserved.
70	46		BIT(1)	qdbqjfprf	Reserved.
70	46	1	BIT(1)	qdbqjvw	Reserved
70	46	2	BIT(1)	qdbqj_type_sup	Join type specified. The type of join is specified in field qdbqj_type.
70	46	3	BIT(5)	qdbqdt_oj	Reserved.
71	47		CHAR(1)	qdbqj_type	Type of join. *J* Inner join *C* Partial outer join *E* Exception join
72	48		CHAR(24)	qdbqdt_28	Reserved.

JREF Join Specification (QDBQ_JREF_T)

JREF Join specification. This structure can be used to define the order in which the files are to be joined. It can also be used to specify any join selection needed to implement the join. Two files (or join results) are specified along with the appropriate join type to be used to join together the two operands. An offset can also be specified to the Selection Specifications (QDBQS) that will define the join criteria that applies to the operands.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0			qdbq_jref_hdr	JREF Join specifications header.
0	0		BIN(4)	qdbq_jref_len	Length of this JREF join specification.
4	4		BIN(2)	qdbq_jref#	Number of JREF Join entries.
6	6		CHAR(10)	qdbqdt_66	Reserved.
16	10		CHAR(*)	qdbq_jref_spec	Start of the JREF Join entries. The structure is defined at QDBQ_JREF_ENTRY_T on "JREF Join Entry (QDBQ_JREF_ENTRY_T)" on page 349.

JREF Join Entry (QDBQ_JREF_ENTRY_T)

JREF Join Entry.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		BIN(2)	qdbq_jref_ entry_type	JREF Join entry type. *0* Join operand *2* Join operator
2	2		CHAR(*)	qdbq_jref_item	JREF Join items. The structure is defined at QDBQ_JREF_OPERAND_T and QDBQ_JREF_OPERATOR_T on "JREF Join Specification (QDBQ_JREF_T)" on page 348.

JREF Join Operand (QDBQ_JREF_OPERAND_T)

JREF Join entry operand.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		BIN(2)	qdbq_jref_file#	JREF Join reference number. The value in this field is used to identify the entry in the QDBQN array associated with this file.
2	2		CHAR(8)	qdbqdt_67	Reserved.

JREF Join Operator (QDBQ_JREF_OPERATOR_T)

JREF Join entry operator.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		BIN(4)	qdbq_jref_jselo	JREF Join entry offset. Offset to the join selection for this JREF Join predicate. The join selection is defined by the Selection Specifications (QDBQS).
4	4		CHAR(1)	qdbq_jref_jtype	JREF Join type. Type of the join specified for this JREF Join predicate. *J* Inner join *C* left partial outer join *E* Exception join
5	5		CHAR(5)	qdbqdt_68	Reserved.

Record Selection Specification (QDBQS_T)

Record selection specification. This structure defines the selection specifications for the files being queried. Selection on the file is done before grouping. If selection is desired on group by results, see structure QDBQGS_T on "Group-by-Selection Specification (QDBQGS_T)" on page 359.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(4)	qdbqsl	Selection specifications length. This is the total length of all selection specifications.
4	4		BIN(2)	qdbqsnum	Number of selection specifications.
6	6		CHAR(10)	qdbqdt_29	Reserved. *(Ref #9.)*
16	10		CHAR(*)	qdbqspec	Start of selection specifications. Displacement to selection item specifications array from structure QDBQS_T (see structure QDBQSIT_T on "Selection Item Specifications (QDBQSIT_T)").

Selection Item Specifications (QDBQSIT_T)

Selection item specifications. This structure is defined at field qdbqspec in structure QDBQS_T (page 350).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(4)	qdbqslen	Selection item length. This length includes the length (QDBQSIT_T) plus the length of the selection item structure.
4	4		BIN(2)	qdbqsitt	Selection item type. *0* Field operand *1* Constant operand *2* Operator *3* Subquery operand *4* Null operand (SAA). This operand is used for `is null` and `is not null` functions. Only equal and not equal operators are allowed. *(Ref #10.)*
6	6		CHAR(*)	qdbqsitm	Selection item. This field is overlaid by the sequence of selection field structures.

Selection Field Operand (QDBQSOPF_T)

Selection field operand. This structure overlays field qdbqsitm in structure QDBQSIT_T (page 350).

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(30)	qdbqsofn	Field name. The field name must be an external name.
30	1E		BIN(2)	qdbqsofj	Field join reference number. 0 indicates that the QDBQR_T format (see "Record Format Specification (QDBQR_T)" on page 346) is searched for the external field name. If the field is not found, the formats of the files in the file specification are searched. If the field name is found in more than one file format, an error is signaled. A value in this field indicates that the external field name is to be found in the file format referenced by using this value as an index into the file list.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
32	20		BIN(2)	qdbqsoqt	Index into the query-definition-template table for the correlated field's associated query definition template. Use zero for noncorrelated fields.
34	22		CHAR(24)	qdbqdt_30	Reserved.

Selection Field Subquery Operand (QDBQSOPS_T)

Selection field subquery operand. This structure overlays field qdbqsitm in structure QDBQSIT_T (page 350)).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(2)	qdbqssub	Index into the query-definition-template offset table for the subquery's query definition template.
2	2		CHAR(1)	qdbqstyp	Subquery operator qualifier. *X'00'* Use the qdbqsop (page 354) field only *X'01'* ALL *X'02'* ANY or SOME Valid values for qdbqsop when qdbqstyp equals 00 are: *Basic predicate* 0001-0006 *Exists* 0045 *In* 0046 Valid values for qdbqsop when qdbqstyp is not equal to 00 are: *Operator* 0001-0006
3	3		CHAR(23)	qdbqdt_31	Reserved.

Selection Constant Operand (QDBQSOPC_T)

Selection Constant Operand. This structure overlays field qdbqsitm in structure QDBQSIT_T (page 350).

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
			CHAR (32793)	qdbqsoch	Constant operand header.
0	0		BIN(4)	qdbqsocl	Constant operand byte length. This only includes the length of the constant in field qdbqsovl (page 354), including apostrophes.
4	4		CHAR(1)	qdbqdt_32	Constant attributes.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
4	4	0	BIT(1)	qdbqsoci	DBCS open constant. *0* This constant is not a DBCS-open literal. *1* This constant is a DBCS-open literal.
4	4	1	BIT(1)	qdbqdt_33	Reserved.
4	4	2	BIT(1)	qdbqsocc	Character constant type. *0* Character string in apostrophes. The character constant is bracketed by apostrophes and any imbedded apostrophes must be represented by two apostrophes. *1* Character string not in apostrophes. The character constant is not bracketed by apostrophes. If it is determined during query processing that the constant should be numeric and if field qdbqsoac in this table is 0, this bit is ignored.
4	4	3	BIT(1)	qdbqsoac	Character constant. *0* Do not assume that this is a character constant. Determination of the type of constant is made during query processing. *1* Assume that this is a character constant.
4	4	4	BIT(1)	qdbqsoco	DBCS-only constant. *0* This constant is not DBCS-only. *1* This constant is DBCS-only.
4	4	5	BIT(1)	qdbqsosr	Special register. *0* This constant operand is not a special register. *1* This constant operand is a special register, defined by the qdbqsorc field.
4	4	6	BIT(1)	qdbqsonl	SAA NULL indicator. *0* This constant operand is not a NULL literal. *1* This constant operand is a NULL literal. The query definition template is synchronized with the format description.
4	4	7	BIT(1)	qdbqdt_34	Reserved.
5	5		CHAR(1)	qdbqsorc	Special register constant. This field is defined by special register constants declared in the record format definition. This field can only be specified if field qdbqsosr is on. *X'01'* User *X'02'* Current date *X'03'* Current time *X'04'* Current timestamp *X'05'* Current time zone *X'06'* Current server

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
6	6		CHAR(1)	qdbqsoft	Date, time, timestamp, format attribute. This field applies only to date, time, or timestamp literals. *X'FE'* Job default *X'FF'* Determine format *X'01'* USA format *X'03'* ISO format *X'05'* EUR format *X'07'* JIS format *X'09'* SAA timestamp *X'17'* MDY format *X'18'* DMY format *X'19'* YMD format *X'1A'* JUL format *X'1B'* HMS format *X'1D'* YYYYNNN format *X'1E'* YYYYMMDDHHMMSS format When the value of this field is X'FF', the format and separator specified in the query-definition-template header (either the qdbqdfmt (page 335) field or the qdbqtfmt (page 335) field, and either the qdbqdsep (page 335) field or the qdbqtsep (page 336) field, for a date or time literal is used first in determining the format and separator of the literal. When the value of this field is X'FE' for a date or time literal, the format and separator are determined using the job attributes. The format value may be X'17', X'18', X'19', X'1A', or X'1B'. The separator specified for qddfdvsp (page 104) is used first in determining the format and separator. When the value of this field is X'FE' for a timestamp literal, the SAA timestamp format is used as the format of the literal.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
7	7		CHAR(1)	qdbqsosp	Date and time separator. This field applies only to date or time literals. It should only be set when the qdbqsoft field is X'FE', X'17', X'18', X'19', X'1A', or X'1B'. *X'00'* Job default separator *X'EE'* Implied separator / Slash separator - Dash separator . Period separator , Comma separator *Blank* Blank separator : Colon separator When the value of this field is X'00', the separator is obtained from the job attributes, which will be one of the preceding values except X'00' or X'EE'. When the value of this field is X'EE', the implied separator for the format is used.
8	8		CHAR(2)	qdbqdt_35	Reserved.
10	A		BIN(2)	qdbqsocd	CCSID value for this literal. If not set to zero, the literal will be tagged with this CCSID. Otherwise, the literal will be tagged with the CCSID specified in the query-definition-template header (see "Query Definition Header (QDBQH_T)" on page 331) or the job default, in that order. This field is only valid for character, DBCS-open, DBCS-only, DBCS-graphic, and UCS-2 literals.
12	C		CHAR(1)	qdbqdt_36	Reserved.
12	C	0	BIT(2)	qdbqdt_37	Reserved.
12	C	2	BIT(1)	qdbqglit	An indicator that the constant is a DBCS-graphic or UCS-2 literal. If this field is a UCS-2 literal, qdbqsocd must be set to a valid UCS-2 CCSID, or qdbqsocd must be zero and qdbqcsdc (see "Query Definition Header (QDBQH_T)" on page 331) must be set to a valid UCS-2 CCSID.
12	C	3	BIT(5)	qdbqdt_38	Reserved.
13	D		CHAR(29)	QDBQDT_39	Reserved. *(Ref #11.)*
42	2A		CHAR (32751)	qdbqsovl	Operand value. The operand value must be in external form.

Selection Operator Item (QDBQSOPR_T)

Selection Operator Item. This structure overlays field qdbqsitm (page 350) in structure QDBQSIT_T.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(2)	qdbqsop	Operators. Relational operators: *X'0001'* Equal *X'0002'* Not equal *X'0003'* Greater than or equal *X'0004'* Less than or equal *X'0005'* Greater than *X'0006'* Less than *X'0007'* Range (inclusive) *X'0041'* Scan *X'0042'* Wildcard scan *X'0043'* Values *X'0045'* Exists *X'0046'* In Boolean operators: *X'000B'* OR *X'000C'* XOR *X'000D'* AND *X'000E'* NOT Case selection operators: *X'0018'* WHEN *X'001B'* ELSE Case operators are only valid when specified as part of a case selection specification.
2	2		CHAR(1)	qdbqswc1	Wildcard value for any single character. This character indicates the value in the character string operand that should be interpreted as matching any single character. This field is only applicable if the qdbqsop field is a wildcard scan.
3	3		CHAR(1)	qdbqswc2	Wildcard value for any number of characters. This character indicates the value in the character string operand that should be interpreted as matching any number of characters. This field is only applicable if the qdbqsop field is a wildcard scan.
4	4		BIN(2)	qdbqvalcnt	Values operand count. This count reflects the number of selection constant operands (values) associated with the values operator. This count must be set if the operator is values and is ignored for all other operators.
6	6		CHAR(1)	qdbqdt_55	Selection operator flags.

Offset		Bit	Type	Variable Name	Field
Dec	**Hex**				
6	6	0	BIT(1)	qdbqescp	Wildcard escape character indicator. This field is valid only for wildcard scan. *0* There is no escape character. *1* There is an escape character specified for the wildcard scan operator by using the third operand.
6	6	1	BIT(1)	qdbqdt_56	Reserved.
6	6	2	BIT(1)	qdbqsopr_ext	Selection operator extension area indicator. *0* Operator extension area (QDBQSOP3_T) does not exist. *1* Operator extension area (QDBQSOP3_T) exists.
6	6	3	BIT(5)	qdbqdt_60	Reserved.
7	7		CHAR(3)	qdbqdt_40	Reserved.
Note: The following fields are not present in a query definition restored from a System/38.					
10	A		CHAR(14)	qdbqsop2	Wildcard value for double-byte characters
10	A		CHAR(2)	qdbqsdbl	Wildcard value for any one double-byte character. This value indicates the value in the DBCS string operand that should be interpreted as matching any one double-byte character. This field is only applicable if field qdbqsop is a wildcard scan and string operand is a DBCS or graphic pattern.
12	C		CHAR(2)	qdbqsdb2	Wildcard value for any number of double-byte characters. This value indicates the value in the double-byte string operand that should be interpreted as matching any number of double-byte or single-byte characters. This field is only applicable if field qdbqsop is a wildcard scan and the string operand is a DBCS or graphic pattern.
14	E		CHAR(3)	qdbqdt_41	Reserved.
17	11		CHAR(2)	qdbqsuo1	Half-width wildcard value for any one double-byte UCS-2 character. This value indicates what value in the UCS-2 operand matches any one double-byte UCS-2 character. This field is only applicable if qdbqsop is a wildcard scan and the pattern is a UCS-2 parameter marker, host variable value, or constant.
19	13		CHAR(2)	qdbqsuo2	Full-width wildcard value for any one double-byte UCS-2 character. This value indicates what value in the UCS-2 operand matches any one double-byte UCS-2 character. This field is only applicable if qdbqsop is a wildcard scan and the pattern is a UCS-2 parameter marker, host variable value, or constant.
21	15		CHAR(2)	qdbqsum1	Half-width wildcard value for any number of double-byte UCS-2 characters. This value indicates what value in the UCS-2 operand matches any number of double-byte UCS-2 characters. This field is only applicable if qdbqsop is a wildcard scan and the pattern is a UCS-2 parameter marker, host variable value, or constant.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
23	17		CHAR(2)	qdbqsum2	Full-width wildcard value for any number of double-byte UCS-2 characters. This value indicates what value in the UCS-2 operand matches any number of double-byte UCS-2 characters. This field is only applicable if qdbqsop is a wildcard scan and the pattern is a UCS-2 parameter marker, host variable value, or constant.
25	19		CHAR(1)	qdbqdt_59	Reserved.

» **Note:** For the wildcard scan operator (qdbqsop=X'0042'), UTF-8 wildcard values should not be specified. If either the match operand, the pattern operand, or the escape character are UTF-8, specify both the EBCDIC equivalents (qdbqswc1, qdbqswc2, qdbqsdb1, and qdbqsdb2) and the UCS-2 equivalents (qdbqsuo1, qdbqsuo2, qdbqsum1, and qdbqsum2). «

Selection Operator Item Extension (QDBQSOP3_T)

Selection Operator Item Extension. This structure overlays field qdbqsitm in structure QDBQSIT_T (page 350) by following QDBQSOPR_T and is only present if qdbqsopr_ext is set to '1'.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(2)	qdbqswc_ccsid	CCSID of wildcard character values that are specified in qdbqswc1, qdbqswc2, qdbqsdb1, and qdbqsdb2. The appropriate associated CCSID is determined depending on the CCSID of the pattern. If needed, this CCSID is used to convert the relevant wildcard characters to the CCSID of the pattern. If set to zero, it is assumed that the wildcard values are in the same CCSID as that of the pattern.
2	2		BIN(2)	qdbqswc_ ccsid_ucs2	CCSID of wildcard character values that are specified in qdbqsuo1, qdbqsuo2, qdbqsum1, and qdbqsum2. If needed, this CCSID is used to convert the relevant wildcard characters to the CCSID of the pattern. If this field is set to 0, it is assumed that the wildcard values are in the same CCSID as the pattern. If this field is specified, it must be a valid UCS-2 CCSID.
4	4		CHAR(28)	qdbqdt_61	Reserved.

Order by Specification (QDBQK_T)

Order by specification. This structure contains a description of how the results of the query should be ordered. Up to 10 000 bytes may be used in ordering.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(16)	qdbqkh	Order by header.
0	0		BIN(2)	qdbqknum	The number of key positions in the order by specifications.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
2	2		CHAR(1)	qdbqkt	Key field ordering type. *U* Unique key fields *D* Duplicate key fields *F* FIFO duplicate key fields *L* LIFO duplicate key fields *C* FCFO duplicate key fields *A, X'00'* Any key field is considered. This field is only used as a guide when considering indexes. Field qdbqopta (page 336)) should be set to on to consider that all indexes build over the query files.
3	3		CHAR(13)	qdbqdt_42	Reserved.
16	10		ARRAY(1) OF CHAR(64)	qdbqkf	Key specifications of 10 000.
16	10		CHAR(30)	qdbqkfld	Key field name. The field name must be an external field name from the QDBQR_T format (unless QDBQR_T is not specified, in which case the field is an external field name from the file format). For the QDBQR_T structure, see "Record Format Specification (QDBQR_T)" on page 346. Field Qddffiob (page 88) must not be X'04' (`neither input nor output`) for a key field.
36	24		CHAR(1)	qdbqksq	Key field sequencing attributes.
36	24	0	BIT(1)	qdbqksad	Ascending or descending sequencing indicator. *0* Ascending sequence *1* Descending sequence
36	24	1	BIT(1)	qdbqdt_43	Reserved.
36	24	2	BIT(1)	qdbqkabs	Absolute value sequence indicator. This bit is ignored for character key fields. *0* Numeric sequence *1* Absolute value sequence
36	24	3	BIT(5)	qdbqdt_44	Reserved.
37	25		CHAR(33)	qdbqdt_45	Reserved.

Group by Specification (QDBQG_T)

Group by specification. This structure contains a description of how the record results of the query should be grouped. All records for which equal values exist in the defined fields are grouped together. Up to 2000 bytes may be used.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(16)	qdbqgh	Group by header.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(2)	qdbqgfnum	The number of group by fields. If the number of group fields is 0, all the records are processed as one group.
2	2		CHAR(14)	qdbqdt_46	Reserved.
16	10		ARRAY(120) OF CHAR(64)	qdbqgf	Group field specification. Up to 120 fields are allowed.
16	10		CHAR(30)	qdbqgfld	Group field name.
46	2E		BIN(2)	qdbqgflj	Field-join reference number. 0 indicates that the QDBQR_T format ("Record Format Specification (QDBQR_T)" on page 346) is searched for the external field name. If the field is not found, the formats of the files in the file specification are searched. If the field name is found in more than one file format, an error is signaled. A value in this field indicates that the external field name is found in the file format referred to by using this value as an index into the file list.
48	30		CHAR(32)	qdbqdt_47	Reserved.

Group-by-Selection Specification (QDBQGS_T)

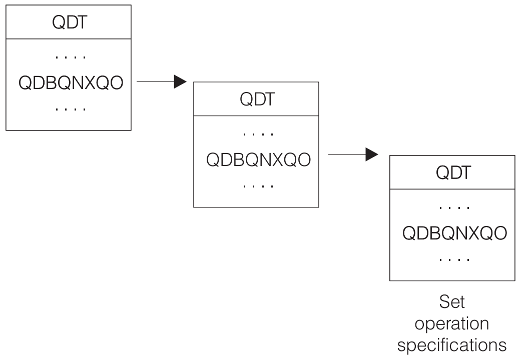
Group-by-selection specification. This structure defines the selection specifications for the group by results. Selection on the group results is performed after the selection on the record is complete and the grouping has been completed.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		CHAR(*)	QDBQGS_T	Group-by-selection specification structure. See "Record Selection Specification (QDBQS_T)" on page 349.
			CHAR(*)	QDBQGSIT_T	The group by selection item specification structure (see "Selection Item Specifications (QDBQSIT_T)" on page 350).

Set Operation Specification (QDBQT_T)

Set operation specification. This structure defines the operation specifications being performed for each set of results generated from each query definition template. These specifications are only valid when more than one query definition template is specified. The set operation specifications must only be specified on the last query definition template.

The specification structure is a stack of operands and operators in reverse notation. Operands are constant literals that identify the relative position of a query definition template among others in the query-definition-template chain. Operators are set operators such as union. For example, given the following query definition templates:



RBAFX588-0

The following operations can be performed:

```
(1st QDT) UNION (2nd QDT) UNION ALL (3rd QDT)
```

The above can be specified in the set operation specification (in reverse notation) as:

```
1 2 UNION 3 UNION ALL
```

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		BIN(4)	qdbqtl	Set specifications length. This is the total length of all set specifications.
4	4		BIN(2)	qdbqtnum	Number of set specifications.
6	6		CHAR(10)	qdbqdt_48	Reserved. *(Ref #13.)*
16	10		CHAR(*)	qdbqtspc	Start of set specifications.

Set Item Specifications (QDBQTIT_T)

Set item specifications. This structure overlays field qdbqtspc (page 360) in structure QDBQT_T.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		BIN(4)	qdbqtlen	Set item length. This length includes the length (QDBQTIT_T) plus the length of the set item structure.
4	4		BIN(2)	qdbqtitt	Set item type. *1* Constant operand *2* Operator *(Ref #14.)*
6	6		CHAR(10)	qdbqtitm	Set item. Use either table QDBQtopC_T or QDBQtopR_T.

Relative Number of Query Definition Template (QDBQtopC_T)

Relative number of query definition template. This structure overlays field qdbqtitm in structure QDBQTIT_T (page 360) .

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
			BIN(2)	qdbqtqdt	Relative number of query definition template.
			CHAR(8)	qdbqdt_49	Reserved.

Set Operators (QDBQtopR_T)

Set operators. This structure overlays field qdbqtitm in structure QDBQTIT_T (page 360) .

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
			CHAR(2)	qdbqtop	Set operators. *X'0001'* Union *X'0002'* Union all
			CHAR(8)	qdbqdt_50	Reserved.

Query Definition Template Offset Table (QDBQQDTS_T)

Query definition template offset table. This structure is set for each unioned outermost query definition template that contains subqueries. This offset table contains offsets for addressability to each query definition template within a union.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		CHAR(16)	qdbqqhdr	Header.
0	0		BIN(2)	qdbqqdtnum	Number of subqueries defined with offsets.
2	2		CHAR(14)	qdbqdt_51	Reserved.
16	10		ARRAY(32) OF CHAR(16)	qdbqqdt	Array of subquery offsets. See structure QDBQQDT_T ("Array of Subquery Offsets (QDBQQDT_T)") for the layout.

Array of Subquery Offsets (QDBQQDT_T)

Array of subquery offsets.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
0	0		BIN(4)	qdbqo	Offset to QDT from start of first QDT in the union.
4	4		CHAR(12)	qdbqdt_52	Reserved.

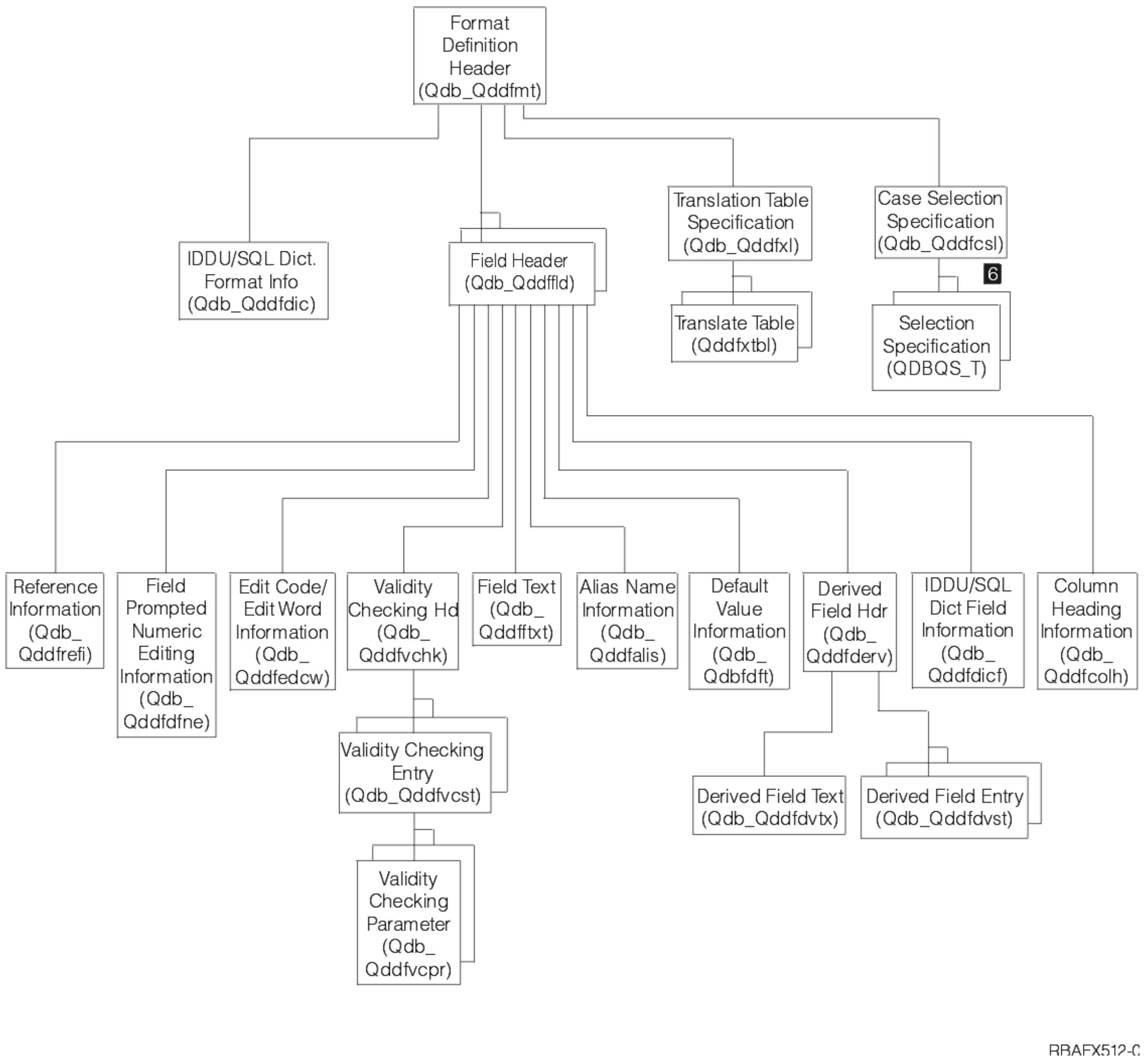
Format Definition Template (Qdb_Qddfmt_t)

The format definition (Qdb_Qddfmt_t) for the QQQQRY API is the same structure that is used by the Retrieve Database File Description (QDBRTVFD) API called FILD0200. Qdb_Qddfmt_t Format (page 362) shows how this information is organized. When more than one entry can appear, the figure indicates this as in **(2)**. For a description of the fields in Qdb_Qddfmt_t and their respective offsets, see "FILD0200 Format (Qdb_Qddfmt Structure)" on page 84 in Retrieve Database File Description (QDBRTVFD) API.

The description and offsets are also in the include source supplied with OS/400. You can see this source in member QDBRTVFD in the QSYSINC library.

The QQQQRY API builds the format definition if it was not created prior to the query.

Qdb_Qddfmt_t Format



Format Definition Header (Qdb_Qddfmt)
IDDU/SQL Dict. Format Info (Qdb_Qddfdic)
Field Header (Qdb_Qddffld)
Translation Table Specification (Qdb_Qddfxl)
Translate Table (Qddfxtbl)
Case Selection Specification (Qdb_Qddfcsl)
6
Selection Specification (QDBQS_T)
Reference Information (Qdb_ Qddfrefi)
Field Prompted Numeric Editing Information (Qdb_ Qddfdfne)
Edit Code/ Edit Word Information (Qdb_ Qddfedcw)
Validity Checking Hd (Qdb_ Qddfvchk)
Field Text (Qdb_ Qddftxt)
Alias Name Information (Qdb_ Qddfalis)
Default Value Information (Qdb_ Qdbfdft)
Derived Field Hdr (Qdb_ Qddfderv)
IDDU/SQL Dict Field Information (Qdb_ Qddfdicf)
Column Heading Information (Qdb_ Qddfcolh)
Validity Checking Entry (Qdb_Qddfvcst)
Validity Checking Parameter (Qdb_ Qddfvcpr)
Derived Field Text (Qdb_Qddfdvtx)
Derived Field Entry (Qdb_Qddfdvst)
RBAFX512-0

User File Control Block (QDBUFCB_T) Structure

User file control block. This structure holds information from the user file control block (UFCB). It contains selected options for the input and output of the specified query.

The options available include:

- Sequence only
- Commitment control
- Block records
- Keyed feedback
- Record length

- Open options
- Release number
- Version number
- Invocation mark count or activation group number
- iSeries system environment
- Null-capable fields
- File dependency
- Level check
- Record format specifications
- Secure
- Shared
- Open scope

In addition, some validity checking is done for this UFCB. CPF4297 is issued if any reserved space in the header of the QDBUFCB_T format is not zero.

The offsets and a description of all the fields contained in this structure are shown in the following table. You can see this source in member QQQQRY in the QSYSINC library.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
0	0		CHAR (1962)	qufcb	Query base UFCB. A character view of the entire user file control block.
0	0		CHAR(174)	reserved1	Reserved.
174	AE		CHAR(1)	shr_secure	Share and secure flags.
174	AE	0	BIT(3)	flglrsva	Reserved.
174	AE	3	BIT(1)	flglshr	Share specified. *0* No type of share was specified on the UFCB. *1* SHARE(YES) or SHARE(NO) was specified on the UFCB.
174	AE	4	BIT(1)	flglshsw	Share value. *0* Not share *1* Share
174	AE	5	BIT(1)	flglsecr	Secure specified.
174	AE	6	BIT(1)	flglud	Secure value. *0* Not secure *1* Secure
174	AE	7	BIT(1)	flglsvb	Reserved.
175	AF		CHAR(1)	open	Open flags.
175	AF	0	BIT(2)	flagrsva	Reserved.
175	AF	2	BIT(1)	flagui	Open input.
175	AF	3	BIT(1)	flaguo	Open output.
175	AF	4	BIT(1)	flaguu	Open update.
175	AF	5	BIT(1)	flagud	Delete.
175	AF	6	BIT(2)	flagsvb	Reserved.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
176	B0		CHAR(4)	relver	Release and version.
176	B0		CHAR(2)	release	Release number. This value must be set to 01.
178	B2		CHAR(2)	version	Version number. This value must be set to 00.
180	B4		BIN(4)	invmkcnt	Mark counter of call or activation group. Set this field to the call mark count when scoping the open to the default activation group. For this case, a `0` indicates a permanent open, and any value greater than 0 indicates a temporary open. Set this field to the activation group mark count when scoping the open to an activation group. **Note:** Setting this field to the default activation group is the same as specifying a call mark of 0 for a permanent open. » This is the low-order four bytes of the mark count value. This field needs to be concatenated with mumkctbh to retrieve the actual invocation mark. «
184	B8		CHAR(1)	markcnt	Mark count and blocked record.
184	B8	0	BIT(1)	flg2mkcp	Mark counter option. *0* The mark counter specified by the invmkcnt field is not used. *1* The mark counter specified by the invmkcnt field is used.
184	B8	1	BIT(1)	flg2rsvl	Reserved.
184	B8	2	BIT(1)	flg2brcd	Blocked records. *0* There are no blocked records. *1* There are blocked records.
184	B8	3	BIT(5)	flg2rsv2	Reserved.
185	B9		CHAR(1)	reserved2	Reserved.
186	BA		CHAR(1)	invact	Mark count usage.
186	BA	0	BIT(1)	flg4rsvl	Reserved.
186	BA	1	BIT(1)	flg4iact	Mark counter usage. *0* The mark counter specified by the invmkcnt field contains a call mark. *1* The mark counter specified by the invmkcnt field contains an activation group number.
186	BA	2	BIT(6)	flg4rsv2	Reserved.
187	BB		CHAR(1)	reserved2a	Reserved.
188	BC		CHAR(1)	native	iSeries environment and process NULLS.
188	BC	0	BIT(2)	flg3rsvl	Reserved.
188	BC	2	BIT(1)	flg3ntve	This field must be set to 1.
188	BC	3	BIT(3)	flg3rsv2	Reserved.
188	BC	6	BIT(1)	flg3null	Process null-capable fields.
188	BC	7	BIT(1)	flg3rsv3	Reserved.
189	BD		CHAR(2)	reserved3a	Reserved.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
191	BF		CHAR(1)	opnscp	Open scope. *A* Open is scoped to the specified activation group, or if this is the default activation group and a call mark is specified, the open is scoped to the program at the call mark specified. *J* Open is scoped to the job. *X'00'* Not specified. The value A is assumed.
192	C0		»CHAR(5)	reserved3b	Reserved.
197	C5		BIN(4)	mumkctbh	High-order four bytes of the mark count value making a total of eight bytes. This needs to be concatenated with invmkcnt to obtain the actual invocation mark.
201	C9		CHAR(7)	reserved3	Reserved.«
Note: The parameter field through the ufcbend field are repeated in the variable-length data area for each parameter.					
208	D0		CHAR(73)	parameter	Variable parameters.
208	D0		BIN(2)	primrlnl	Primary record length. Initialize to -1 to deactivate.
210	D2		BIN(2)	primrlnv	The user-specified record length.
212	D4		BIN(2)	filedep	File-dependent I/O. Initialize to -3 to deactivate.
214	D6	0	BIT(1)	fildonoff	File-dependent option. *On* This is file dependent. *Off* This is not file dependent.
214	D6	1	BIT(7)	fldrsvl	Reserved.
215	D7		BIN(2)	lvlchk	Level-check option. Initialize to -6 to deactivate.
217	D9	0	BIT(1)	lvlonoff	Level-check option. *On* Perform level checking *Off* Do not perform level checking
217	D9	1	BIT(7)	lvlrsvl	Reserved.
218	DA		BIN(2)	recfmts	Record format sequence numbers for level checking.
220	DC		BIN(2)	maximum	The maximum number of formats.
222	DE		BIN(2)	curnum	The current number of formats.
224	E0		ARRAY(75) of CHAR(23)	formats	Array of format names and sequence numbers
224	E0		CHAR(10)	name	The format name.
234	EA		CHAR(13)	number	The format sequence number.
1949	79D		BIN(2)	keyfdbk	Key feedback. Initialize to -53 to deactivate.
1951	79F	0	BIT(1)	keyonoff	Key feedback option. *On* Provide feedback *Off* Do not provide feedback
1951	79F	1	BIT(7)	keyrsvl	Reserved.
1952	7A0		BIN(2)	seqonly	Sequential processing. Initialize to -58 to deactivate.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
1954	7A2	0	BIT(1)	seqonoff	Sequential processing option. *On* Use Fast sequence processing *Off* Use standard sequence processing
1954	7A2	1	BIT(1)	numonoff	Fast sequential processing option. *On* Number of records to transfer to or from the I/O buffers for fast sequential processing is specified. *Off* The number of records to transfer to or from the I/O buffers is not specified.
1954	7A2	2	BIT(6)	seqrsvl	Reserved.
1955	7A3		BIN(2)	numrecs	The number of records to transfer to or from the I/O buffers for fast sequential processing.
1957	7A5		BIN(2)	commitc	Commitment control. Initialize to -59 to deactivate.
1959	7A7		CHAR(1)	control	Commitment control and optional record-locking level. Possible values are: *X'00'* Do not place the member under commitment control when it is opened. This would be the same as specifying the Start Commitment Control command as STRCMTCLT COMMIT(*NO). *X'80'* Place the member under commitment control when it is opened, and use the record-locking level default used on the Start Commitment Control command, that is, STRCMTCTL COMMIT (*YES). *X'82'* Place the member under commitment control when it is opened and use record-locking level *CHG, that is, STRCMTCTL COMMIT (*YES,*CHG). *X'86'* Place the member under commitment control when it is opened and use record-locking level *CS, that is, COMMIT *YES,*CS). *X'87'* Place the member under commitment control when it is opened and use record-locking level *ALL, that is, COMMIT (*YES,*ALL).
1960	7A8		BIN(2)	ufcbend	This field must be set to 32767, the end of the variable area parameters. Set this field to ENDLIST.
1962	7AA		BIN(4)	dummy	Dummy pointer to force boundary alignment for the user file control block structure.

Value for Query Variable Fields (QQQVALS_T) Structure

The structure is used to supply the values for the variable fields used by the QQQQRY API. The offsets and a description of all the fields contained in this structure are shown in the following table. You can see this source in member QQQQRY in the QSYSINC library.

Offset		Bit	Type	Variable Name	Field
Dec	Hex				
0	0		BIN(2)	qqqvvalnum	Number of values in list.

Offset					
Dec	Hex	Bit	Type	Variable Name	Field
2	2		CHAR(30)	qqqvals_l	Reserved.
32	20		ARRAY(1) OF CHAR(48)	qqqvlst	List of variable field values referenced by field Qddfvarx. See field Qddfvarx (page 90) for the Qddfvarx field. *(Ref #15.)*
32	20		Pointer	qqqvsptr	Space pointer to the host variable value. The value must be in internal form.
48	30		CHAR(7)	qqqvattr	Attributes of value.
48	30	0	CHAR(1)	qqqvatyp	Scalar type *X'00'* Binary *X'01'* Floating point *X'02'* Zoned decimal *X'03'* Packed *X'04'* Character *X'06'* Graphic *X'07'* DBCS-only *X'08'* DBCS-either *X'09'* DBCS-open *X'0B'* Date *X'0C'* Time *X'0D'* Timestamp
49	31		CHAR(2)	qqqvalen	Scalar length for character, binary, floating point, only, either, or open. For graphic, this is also the number of bytes (not characters).
49	31		CHAR(1)	qqqvadec	Fractional digits for zoned or packed.
50	32		CHAR(1)	qqqvatot	Total digits for zoned or packed.
51	33		CHAR(4)	qqqvaary	Container for precision and digits for binary values.
51	33		CHAR(1)	qqqvbind	Fractional digits for binary value.
52	34		CHAR(1)	qqqvbint	Total digits for binary value.
53	35		CHAR(2)	qqqvals_2	Reserved.
55	37		CHAR(1)	qqqvals_3	Field attributes.
55	37	0	BIT(1)	qqqvvlen	Variable length host variable field. *0* The host variable field is not variable length. *1* The host variable field is variable length.
55	37	1	BIT(1)	qqqvnulll	The form of field qqqvsptr (page 367). If on, qqqvsptr is ignored and the literal is the null value. If off, the literal pointed to by qqqvsptr is used.
55	37	2	BIT(1)	qqqvzerol	The length of field qqqvsptr. If on, qqqvsptr is ignored and the literal is zero length. If off, the literal pointed to by qqqvsptr is used.
55	37	3	BIT(5)	qqqvals_4	Reserved.

Offset					
Dec	**Hex**	**Bit**	**Type**	**Variable Name**	**Field**
56	38		CHAR(1)	qqqvdvft	Date, time, and timestamp format attribute. This field applies to date, time, or timestamp values only, where the field qqqvatyp in this structure is date, time, or timestamp. *X'00'* Job default *X'FF'* Determine format *X'01'* USA format *X'03'* ISO format *X'05'* EUR format *X'07'* JIS format *X'09'* SAA timestamp *X'17'* MDY format *X'18'* DMY format *X'19'* YMD format *X'1A'* JUL format *X'1B'* HMS format *X'1D'* YYYY NNN format *X'1E'* YYYY MM DDDD HH MM SS format These formats are optional. If the value is X'FF', the format is in the query definition template header and that format is used first in determining the format. See field qdbqdfmt (page 335) or field qdbqtfmt (page 335) if the format is in the query definition template header.
57	39		CHAR(1)	qqqvdvsp	Date, time, and timestamp separator. This field is only set when field qqqvdvft in this structure is X'17', X'18', X'19', X'1A', or X'1B'.
58	3A		BIN(2)	qqqvcsid	CCSID of value.
59	3C		CHAR(20)	qqqvals_5	Reserved.

Usage Notes

In multithreaded jobs, this command is not threadsafe for distributed files and fails for distributed files that use relational databases of type *SNA. This command also is not threadsafe and fails for Distributred Data Management (DDM) files of type *SNA.

Error Messages

Message ID	Error Message Text
CPF2114 E	Cannot allocate object &1 in &2 type *&3.
CPF2115 E	Object &1 in &2 type *&3 damaged.
CPF2169 E	Job's sort sequence information not available.
CPF24B4 E	Severe error while addressing parameter list.
CPF2619 E	Table &1 not found.
CPF3BCC E	Language identifier &1 not valid.
CPF3BC6 E	Sort sequence &1 not valid.
CPF3BC7 E	CCSID &1 outside of valid range.

Message ID	Error Message Text
CPF3BC8 E	Conversion from CCSID &1 to CCISID &2 is not supported.
CPF3BC9 E	Conversion from CCSID &1 to CCISID &2 is not defined.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter is not valid.
CPF3FC0 E	Language identifier is not valid.
CPF4000 E	All CPF40xx messages could be returned. xx is from 01 to FF.
CPF4100 E	All CPF41xx messages could be returned. xx is from 01 to FF.
CPF4200 E	All CPF42xx messages could be returned. xx is from 01 to FF.
CPF4300 E	All CPF43xx messages could be returned. xx is from 01 to FF.
CPF5000 E	All CPF50xx messages could be returned. xx is from 01 to FF.
CPF5100 E	All CPF51xx messages could be returned. xx is from 01 to FF.
CPF5200 E	All CPF52xx messages could be returned. xx is from 01 to FF.
CPF5300 E	All CPF53xx messages could be returned. xx is from 01 to FF.
CPF8133 E	Table &4 in &9 damaged.
CPF9800 E	All CPF98xx messages could be signaled. xx is from 01 to FF.

Example

See Code disclaimer information for information pertaining to code examples.

For examples that use the QQQQRY API, see Defining Queries in the Examples: APIs topic.

API introduced: V2R2

Top | "Database and File APIs," on page 1 | APIs by category

Run Database Hash (QDBRUNHA) API

Required Parameter Group:

1	Hash name	
Input	Char(10)	
2	Function	
Input	Char(1)	
3	Number of keys	
Input	Binary(4)	
4	Key values	
Input	Char(*)	
5	Data	
I/O	Char(*)	
6	Length of Data	
Input	Binary(4)	
7	Return code	
Output	Binary(4)	

Default Public Authority: *USE

Threadsafe: No

The Run Database Hash (QDBRUNHA) API allows the user to FETCH, UPDATE, DELETE and INSERT data into existing database files using an alternative access method. The hash approach can be used on relatively static files in situations where it is desirable to reduce the amount of memory that is consumed by indexes. Its affectiveness is reduced in memory-rich environments or environments with dynamic data.

Authorities and Locks

HASH User Space in Library QUSRSYS
*OBJOPR, *READ, and *UPDATE

Required Parameter Group

Hash name
INPUT; CHAR(10)

The hash name used to access the data. See the description of the Create Database Hash (QCreateDatabaseHash) API for information on defining and naming the hash.

Function
INPUT; CHAR(1)

The function to be performed. The possible values are character digits as follows:

1	Fetch a row for read purposes only
2	Fetch a row with intent to update

3	Update the currently locked row
4	Delete the currently locked row
5	Insert a row into the database

Number of keys
INPUT; BINARY(4)

The number of keys used by the hash.

Key values
INPUT; CHAR(*)

A structure containing up to five rows of the name of key fields and the value of key fields in the order that they appear in the logical file used to create the hash. For more details, see "Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
		CHAR(10)	Name of key
		Binary(4)	Value of key

Data I/O; CHAR(*)

A pointer to the actual row of data from the database to be manipulated according to the function parameter. The row will be inserted, deleted, updated, or fetched from the database.

Data length
INPUT; BINARY(4)

The length of the buffer in the application that will receive or contain the data.

Return code
OUTPUT; BINARY(4)

A numeric indication as to what took place during the hash function request. The possible values are:

0	No errors.
1	Hash user space does not exist.
2	Hash does not exist.
100	Record not found.
812	Lock-wait time-out.
-99	Another error was encountered and ignored. See job log for details.

Field Descriptions

Name of key. The name of the key field in the order that they appear in the logical file referenced in the associated Create Database Hash (QDBCRTHA) API. The names are for documentation purposes only.

Value of key. The key value used by the API to access the appropriate record in the file.

Error Messages

Only the error conditions listed in the return code parameter are monitored. No error messages other than the value of the return code parameter are returned.

API Introduced: V4R3

Top | "Database and File APIs," on page 1 | APIs by category

Database Performance APIs

The Database Performance APIs are:

- "Cancel Requested Statistics Collections (QDBSTCRS, QdbstCancelRequestedStatistics) API" on page 373 (QDBSTCRS, QdbstCancelRequestedStatistics) cancels statistics collections that have been requested, but are not yet completed or not successfully completed.
- "Clear SQL Database Monitor Statistics (QQQCSDBM) API" on page 379 (QQQCSDBM) clears and frees the associated memory area of the database monitor statistics.
- "Delete Statistics Collections (QDBSTDS, QdbstDeleteStatistics) API" on page 381 (QDBSTDS, QdbstDeleteStatistics) deletes existing completed statistics collections immediately.
- "Dump SQL Database Monitor (QQQDSDBM) API" on page 387 (QQQDSDBM) dumps the SQL database monitor that has been gathered.
- "End SQL Database Monitor (QQQESDBM) API" on page 390 (QQQESDBM) ends the memory-based SQL database monitor.
- "List Requested Statistics Collections (QDBSTLRS, QdbstListRequestedStatistics) API" on page 392 (QDBSTLRS, QdbstListRequestedStatistics) lists all of the columns and combination of columns and file members that have background statistic collections requested, but not yet completed.
- "List Statistics Collection Details (QDBSTLDS, QdbstListDetailStatistics) API" on page 401 (QDBSTLDS, QdbstListDetailStatistics) lists additional statistics data for a single statistics collection not returned by the List Statistics Collections (QDBSTLS, QdbstListStatistics) API.
- "List Statistics Collections (QDBSTLS, QdbstListStatistics) API" on page 412 (QDBSTLS, QdbstListStatistics) lists all of the columns and combination of columns for a given file member that have statistics available.
- "Query SQL Database Monitor (QQQQSDBM) API" on page 424 (QQQQSDBM) returns information about the activity of the SQL and the original database monitor.
- "Request Statistics Collections (QDBSTRS, QdbstRequestStatistics) API" on page 426 (QDBSTRS, QdbstRequestStatistics) requests that one or more statistics collections for a given set of columns of a database file member be created.
- "Start SQL Database Monitor (QQQSSDBM) API" on page 434 (QQQSSDBM) starts the memory-based SQL database monitor.
- "Update Statistics Collections (QDBSTUS, QdbstUpdateStatistics) API" on page 437 (QDBSTUS, QdbstUpdateStatistics) updates the attributes and refreshes the data of an existing single statistics collection.
- "Visual Explain (QQQVEXPL) API" on page 444 (QQQVEXPL) is used to create a query graph that graphically displays the execution of an SQL statement

«

Top | Database and File APIs | APIs by category

Cancel Requested Statistics Collections (QDBSTCRS, QdbstCancelRequestedStatistics) API

Required Parameter Group:

1	Input data	
Input	Char(*)	
2	Length of input data	
Input	Binary(4)	
3	Format of input data	
Input	Char(8)	
4	Feedback area	
Output	Char(*)	
5	Length of feedback area	
Input	Binary(4)	
6	Feedback keys	
Input	Array(*) of Binary(4)	
7	Number of feedback keys	
Input	Binary(4)	
8	Error code	
I/O	Char(*)	

Service Program Name: QDBSTMGR

Default Public Authority: *USE

Threadsafe: Yes

The Cancel Requested Statistics Collections (QDBSTCRS, QdbstCancelRequestedStatistics) API allows the user to cancel requested, but not yet completed or not successfully completed statistics collections, and also provides an option to restart a single cancelled request.

Section overview

Authorities and Locks

ASP Device Authority
*EXECUTE

File Authority
*OBJALTER, *OBJOPR

File Library Authority
*EXECUTE

File Lock
*SHRRD

Required Parameter Group

Input data
INPUT; CHAR(*)

The buffer containing the input parameters according to the format of input data parameter. The buffer content has to start at a 4-byte boundary.

Length of input data
INPUT; BINARY(4)

The length of the input data buffer provided.

Format of input data
INPUT; CHAR(8)

The format of the input data. Possible values are:

STIC0100	Cancel a single requested statistics collection using an internal request ID.
STIC0200	Cancel all requested statistics collections for a single database file member.

Refer to “STIC0100 Input Format” on page 375 and “STIC0200 Input Format” on page 375 for more information.

Feedback area
OUTPUT; CHAR(*)

The buffer to receive feedback data. See “Feedback Area Format” on page 375 for more information. The buffer content has to start at a 4-byte boundary.

Length of feedback area
INPUT; BINARY(4)

The length of the feedback area buffer provided. The required minimum length is 16, to fit the feedback area header (see “Feedback Area Format” on page 375).

Feedback keys
INPUT; ARRAY(*) OF BINARY(4)

The list of fields to return in the feedback area. For a list of valid keys, see “Valid Keys - Feedback” on page 375.

Number of feedback keys
INPUT; BINARY(4)

The number of fields to return in the feedback area. If 0 is specified, all other feedback area parameters (*Feedback area*, *Length of feedback area*, and *Feedback keys*) are ignored.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

STIC0100 Input Format

Use this format to cancel the single statistics collection uniquely identified by an internal request ID. See "Field Descriptions" on page 376 for details of the fields listed.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(16)	Internal request ID
16	10	CHAR(12)	Restart option
28	1C	CHAR(*)	Reserved

STIC0200 Input Format

Use this format to cancel all not yet completed statistics collection for a given file member. See "Field Descriptions" on page 376 for details of the fields listed.

Note: When using this option mainly to target system initiated requests, it is recommended that you first use the blocking function provided by the "Update Statistics Collections (QDBSTUS, QdbstUpdateStatistics) API" on page 437 (QDBSTUS, QdbstUpdateStatistics) API to prevent the system from issuing the requests again.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	ASP device name
10	A	CHAR(10)	File name
20	14	CHAR(10)	File library name
30	1E	CHAR(10)	File member name
40	28	CHAR(*)	Reserved

Valid Keys - Feedback

Use the following keys to specify the fields to be returned in the feedback area. Each key can only be specified once. See "Field Descriptions" on page 376 for details of the fields listed.

Key	Type	Description
1	CHAR(10)	ASP device name used
2	CHAR(10)	File name used
3	CHAR(10)	File library name used
4	CHAR(10)	File member name used
21	BINARY(4)	Number of statistics collection requests cancelled.

Feedback Area Format

The fields returned in the feedback area will be returned in the order requested. See "Field Descriptions" on page 376 for details of the fields listed.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	BINARY(4)	Number of key fields returned
12	C	BINARY(4)	Number of key fields available
These fields repeat, in the order listed, for each key selected.		BINARY(4)	Length of field information returned
		BINARY(4)	Key identifier
		BINARY(4)	Length of data
		CHAR(*)	Data
		CHAR(*)	Reserved (padding to the next 4-byte boundary)

Field Descriptions

ASP device name. The name of one auxiliary storage pool (ASP) device in the ASP group in which the library and file are located. The ASP device must have a status of 'Available'. The documented authority is required for the given ASP and the primary of the corresponding ASP group. The name can be a specific ASP device name (for an ASP with a number greater than 32), or one of the following special values:

*	Locate the library and file in the name space for the current thread.
SYSBAS	Locate the library and file in the system ASP (ASP number 1) and all basic ASPs (ASP numbers 2 through 32).

ASP device name used. The actual auxiliary storage pool device name used, after possible resolution of special values.

Data. The data returned for the key identifier.

File library name. Where the file for which statistics collections are to be cancelled is located. You can use these special values for the library name, if the *ASP Device Name* is *:

CURLIB	The job's current library or QGPL if the current library is not set.
LIBL	The library list.
USRLIBL	Libraries listed in the user portion of the library list.

File library name used. The actual file library name used, after possible resolution of special values.

File member name. The name of the file member to be used for the cancel request. This value can be a specific file member name or one of the following special values:

FIRST	The first member (in the order created) in the specified file.
LAST	The last member (in the order created) in the specified file.

File member name used. The actual file member name used, after possible resolution of special values.

File name. The name of the file for which statistics collections are to be cancelled. The file has to be an existing local, single format, physical file.

File name used. The actual file name used.

Internal request ID. Uniquely identifies statistics collections requested earlier. This ID can be obtained from one of the following APIs:
- "Request Statistics Collections (QDBSTRS, QdbstRequestStatistics) API" on page 426 (QDBSTRS, QdbstRequestStatistics) API,
- "List Requested Statistics Collections (QDBSTLRS, QdbstListRequestedStatistics) API" on page 392 QDBSTLRS, QdbstListRequestedStatistics) API, or
- "Update Statistics Collections (QDBSTUS, QdbstUpdateStatistics) API" on page 437 (QDBSTUS, QdbstUpdateStatistics) API.

Key identifier. The field returned. For a list of valid keys see "Valid Keys - Feedback" on page 375.

Length of data. The length of the data returned for the field.

Length of field information returned. Total number of bytes returned for this field in the feedback area.

Number of key fields available. Number of fields that could be returned in the feedback area.

Number of key fields returned. Number of fields returned in the feedback area.

Number of bytes available. Number of bytes that could be returned in the feedback area.

Number of bytes returned. Number of bytes returned in the feedback area.

Number of key fields available. Number of fields that could be returned in the feedback area.

Number of key fields returned. Number of fields returned in the feedback area.

Number of statistics collection requests cancelled. Number of statistics collection requests actually cancelled .

Reserved. Reserved for future use. If this field is input, the field must be set to hexadecimal zeros.

Reserved (in feedback area format). Structure padding to guarantee alignment to the next four-byte boundary.

Restart option. Allows the cancelled request to optionally be restarted. The possible values are:

**NONE*	Do not restart, but just cancel the request, if its in active state, or remove the request, if its in pending or error state. **Note:** When using this option for system-initiated requests, it is recommended that you first use the blocking function provided by the "Update Statistics Collections (QDBSTUS, QdbstUpdateStatistics) API" on page 437 (QDBSTUS, QdbstUpdateStatistics) API to prevent the system from issuing the request again.
**IMMEDIATE*	Restart the request immediately. The statistics collection will run in the user's process. Control will not return to the API invoker until the collection is complete. **Note:** If the request has an active status and not a pending or error status, the cancel request will be ignored.

Error Messages

Message ID	Error Message Text
CPF0623 E	Field &1 not found in record format &2.
CPF1866 E	Value &1 for number of fields to return not valid.
CPF2105 E	Object &1 in &2 type *&3 not found.
CPF2113 E	Cannot allocate library &1.
CPF2173 E	Value for ASPDEV not valid with special value for library.
CPF218C E	&1 not a primary or secondary ASP.
CPF3141 E	Member &2 not found.
CPF34C0 E	Value &1 for number of fields to return parameter not valid.
CPF3C07 E	Error occurred while retrieving information from object &1.
CPF3C1D E	Length specified in parameter &1 not valid.
CPF3C21 E	Format name &1 is not valid.
CPF3C29 E	Object name &1 is not valid.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C39 E	Value for reserved field not valid.
CPF3C3B E	Value for parameter &2 for API &1 not valid.
CPF3C82 E	Key &1 not valid for API &2.
CPF3C89 E	Key &1 specified more than once.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF4268 E	Object &1 in &2 type *&3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9804 E	Object &2 in library &3 damaged.
CPF980B E	Object &1 in library &2 not available.
CPF9810 E	Library &1 not found.
CPF9812 E	File &1 in library &2 not found.
CPF9814 E	Device &1 not found.
CPF9820 E	Not authorized to use library &1.
CPF9822 E	Not authorized to file &1 in library &2.
CPF9825 E	Not authorized to device &1.
CPF9826 E	Cannot allocate file &2.
CPF9830 E	Cannot assign library &1
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB401 E	API &3 failed with reason code &1.
CPFB8ED E	Device description &1 not correct for operation.

Related Information

- the <**qdbst.h**> include file in library QSYSINC, for API-related structure declarations and special value declarations
- the <**qdbstmgr.h**> include file in library QSYSINC, for the QdbstCancelRequestedStatistics API prototype
- the <**qdbstcrs.h**> include file in library QSYSINC, for the QDBSTCRS API prototype
- "Delete Statistics Collections (QDBSTDS, QdbstDeleteStatistics) API" on page 381 (QDBSTDS, QdbstDeleteStatistics) API
- "List Requested Statistics Collections (QDBSTLRS, QdbstListRequestedStatistics) API" on page 392 (QDBSTLRS, QdbstListRequestedStatistics) API
- "List Statistics Collection Details (QDBSTLDS, QdbstListDetailStatistics) API" on page 401 (QDBSTLDS, QdbstListDetailStatistics) API

- "List Statistics Collections (QDBSTLS, QdbstListStatistics) API" on page 412 (QDBSTLS, QdbstListStatistics) API
- "Request Statistics Collections (QDBSTRS, QdbstRequestStatistics) API" on page 426 (QDBSTRS, QdbstRequestStatistics) API
- "Update Statistics Collections (QDBSTUS, QdbstUpdateStatistics) API" on page 437 (QDBSTUS, QdbstUpdateStatistics) API

API introduced: V5R2

Top | "Database and File APIs," on page 1 | APIs by category

Clear SQL Database Monitor Statistics (QQQCSDBM) API

Required Parameter Group:

1	Memory handle to clear
Input	Char(10)
2	Job or memory handle name
Input	Char(26)
3	Error code
I/O	Char(*)

Default Public Authority: *USE

Threadsafe: Yes

The Clear SQL Database Monitor Statistics (QQQCSDBM) API clears and frees the associated memory area of the database monitor statistics. Associated APIs include the following:

- Dump SQL Database Monitor (QQQDSDBM)
- End SQL Database Monitor (QQQESDBM)
- Query SQL Database Monitor (QQQQSDBM)
- Start SQL Database Monitor (QQQSSDBM)

Authorities and Locks

Current User Profile
*JOBCTL

Required Parameter Group

Memory area to clear
INPUT; CHAR(10)

The memory area to be cleared or freed. The possible values are:

**ALL*	Clear the monitor data associated with the *ALL monitor (the monitor started against ALL jobs). Memory areas associated with QQQSSDBM started on individual jobs will not be cleared. No storage is freed.
**NAMED*	Clear the memory handle specified by the job or memory handle name parameter (and matches the name of a memory handle specified on the QQQSSDBM API). No storage is freed. Only the first 6 characters will be used for naming the memory handle.

**JOB*	Clear the job specific data associated with the job name specified in the job or memory handle name parameter. No storage is freed.
**RESET*	Clear and free all memory associated with all active or inactive database monitors. The *RESET option cannot be specified on a specific job or memory handle.

Job or memory handle name

INPUT; CHAR(26)

This parameter depends on the value specified for the memory area to clear parameter. If the value is:

**ALL*	This parameter must be set to blanks.
**RESET*	This parameter must be set to blanks.
**NAMED*	The CHAR(10) name of a memory handle whose data is to be cleared. Only the first 6 characters will be used for naming the memory handle, with the remaining characters set to blanks.
**JOB*	The CHAR(26) qualified job name of a job-specific monitor to dump. The qualified job name has three parts:

Job Name

CHAR(10). A specific job name, a generic name, or one of the following special values:

** or *CURRENT*

Only the job that this program is running in. The rest of the qualified job name parameter must be blank.

**ALL* All jobs. The rest of the job name parameter must be blank.

User Name

CHAR(10). A specific user profile name.

Job Number

CHAR(6). A specific job number.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	Error Message Text
CPD0172 D	Error Parameters passed on CALL do ot match those required.
CPF222E E	&1 special authority required.
CPF3CF1 E	Error code parameter not valid.

API introduced: V4R3

Top | "Database and File APIs," on page 1 | APIs by category

Delete Statistics Collections (QDBSTDS, QdbstDeleteStatistics) API

Required Parameter Group:

1	Input data	**Input**	Char(*)
2	Length of input data	**Input**	Binary(4)
3	Format of input data	**Input**	Char(8)
4	Feedback area	**Output**	Char(*)
5	Length of feedback area	**Input**	Binary(4)
6	Feedback keys	**Input**	Array(*) of Binary(4)
7	Number of feedback keys	**Input**	Binary(4)
8	Error code	**I/O**	Char(*)

Service Program Name: QDBSTMGR

Default Public Authority: *USE

Threadsafe: Yes

The Delete Statistics Collections (QDBSTDS, QdbstDeleteStatistics) API deletes existing, completed statistics collections for a database file member.

Section overview

Authorities and Locks

ASP Device Authority
    *EXECUTE

File Authority
*OBJALTER, *OBJOPR

File Library Authority
*EXECUTE

File Lock
*SHRRD

Required Parameter Group

Input data
INPUT; CHAR(*)

The buffer containing the input parameters according to the format of input data parameter. The buffer content has to start at a 4-byte boundary.

Length of input data
INPUT; BINARY(4)

The length of the input data buffer provided.

Format of input data
INPUT; CHAR(8)

The format of the input data. Possible values are:

STID0100	Delete Statistics Collections input parameters using internal statistics ID.
STID0200	Delete Statistics Collections input parameters using detailed statistics description.

Refer to "STID0100 Input Format" on page 383 and "STID0200 Input Format" on page 383 for more information.

Feedback area
OUTPUT; CHAR(*)

The buffer to receive feedback data. See "Feedback Area Format" on page 384 for more information. The buffer content has to start at a 4-byte boundary.

Length of feedback area
INPUT; BINARY(4)

The length of the feedback area buffer provided. The required minimum length is 16, to fit the feedback area header (see "Feedback Area Format" on page 384).

Feedback keys
INPUT; ARRAY(*) OF BINARY(4)

The list of fields to return in the feedback area. For a list of valid keys, see "Valid Keys - Feedback" on page 383.

Number of feedback keys
INPUT; BINARY(4)

The number of fields to return in the feedback area. If zero is specified, all other feedback area parameters (*Feedback area*, *Length of feedback area*, and *Feedback keys*) are ignored.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

STID0100 Input Format

Delete statistics collections input parameters using internal statistics ID. See “Field Descriptions” on page 384 for details of the fields listed.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	ASP device name
10	A	CHAR(10)	File name
20	14	CHAR(10)	File library name
30	1E	CHAR(10)	File member name
40	28	CHAR(16)	Internal statistics ID
56	38	CHAR(*)	Reserved

STID0200 Input Format

Delete statistics collections input parameters using detailed statistics description. See “Field Descriptions” on page 384 for details of the fields listed.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	ASP device name
10	A	CHAR(10)	File name
20	14	CHAR(10)	File library name
30	1E	CHAR(10)	File member name
40	28	BINARY(4)	Offset to columns
44	2C	BINARY(4)	Number of columns
48	30	CHAR(*)	Reserved
These fields repeat, in the order listed, for each column, starting at the given offset.		BINARY(4)	Length of column definition
		CHAR(10)	Column name
		CHAR(10)	Translation table name
		CHAR(10)	Translation table library name
		CHAR(2)	Reserved
		CHAR(*)	Reserved

Valid Keys - Feedback

Use the following keys to specify the fields to be returned in the feedback area. Each key can only be specified once. See “Field Descriptions” on page 384 for details of the fields listed.

Key	Type	Description
1	CHAR(10)	ASP device name used
3	CHAR(10)	File library name used
4	CHAR(10)	File member name used
20	BINARY(4)	Number of statistics collections deleted.

Feedback Area Format

The fields returned in the feedback area are returned in the order requested. See "Field Descriptions" for details of the fields listed.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	BINARY(4)	Number of key fields returned
12	C	BINARY(4)	Number of key fields available
These fields repeat, in the order listed, for each key selected.		BINARY(4)	Length of field information returned
		BINARY(4)	Key identifier
		BINARY(4)	Length of data
		CHAR(*)	Data
		CHAR(*)	Reserved (padding to the next 4-byte boundary)

Field Descriptions

ASP device name. The name of one auxiliary storage pool (ASP) device in the ASP group in which the library and file are located. The ASP device must have a status of 'Available'. The documented authority is required for the given ASP and the primary of the corresponding ASP group. The name can be a specific ASP device name (for an ASP with a number greater than 32), or one of the following special values:

*	Locate the library and file in the name space for the current thread.
**SYSBAS*	Locate the library and file in the system ASP (ASP number 1) and all basic ASPs (ASP numbers 2 through 32).

ASP device name used. The actual auxiliary storage pool device name used, after possible resolution of special values.

Column name. The name of a single column within the statistics collection definition.

Data.; The data returned for the key identifier.

File library name. Where the file for which statistics collections are to be deleted is located. You can use these special values for the library name, if the *ASP Device Name* is *:

**CURLIB*	The job's current library or QGPL if the current library is not set.
**LIBL*	The library list.
**USRLIBL*	Libraries listed in the user portion of the library list.

File library name used. The actual file library name used, after possible resolution of special values.

File member name. The name of the file member to be used for the delete statistics collections request. This value can be a specific file member name or one of the following special values:

**FIRST*	The first member (in the order created) in the specified file.
**LAST*	The last member (in the order created) in the specified file.

File member name used. The actual file member name used, after possible resolution of special values.

File name. The name of the file for which statistics collections are to be deleted. The file has to be an existing local, single format, physical file.

Internal statistics ID. Together with the qualified file name and member name, this represents a unique ID for the statistics collection to be deleted. See “List Statistics Collections (QDBSTLS, QdbstListStatistics) API” on page 412 (QDBSTLS, QdbstListStatistics) API for additional information.

Note: If set to all X’00’, all statistics collections for the file member are deleted.

Key identifier. The field returned. For a list of valid keys see “Valid Keys - Feedback” on page 383.

Length of column definition. The length of this column definition.

Length of data. The length of the data returned for the field.

Length of field information returned. The total number of bytes returned for this field in the feedback area.

Number of bytes available. The number of bytes that could be returned in the feedback area.

Number of bytes returned. The number of bytes returned in the feedback area.

Number of columns. The number of columns within the single statistics collection definition. The maximum value for this number is 1. If set to zero, all statistics collections for the file member are deleted.

Number of key fields available. The number of fields that can be returned in the feedback area.

Number of key fields returned. The number of fields returned in the feedback area.

Number of statistics collections deleted. The number of actually deleted statistics collections.

Offset to columns. The offset to the start of the list of column definitions.

Reserved. Reserved for future use. If this field is input, the field must be set to hexadecimal zeros.

Reserved (in feedback area format). Structure padding to guarantee alignment to the next four-byte boundary.

Translation table name. The name of the translation table that was specified, when the statistics collection was requested. The translation table does not necessarily have to exist anymore. The name is used for identification purposes only.

Translation table library name. The actual name of the translation table library that was used, when the statistics collection was requested. The translation table library does not necessarily have to exist anymore. The name is used for identification purposes only.

Error Messages

Message ID	**Error Message Text**
CPF0623 E	Field &1 not found in record format &2.
CPF1866 E	Value &1 for number of fields to return not valid.

Message ID	Error Message Text
CPF2105 E	Object &1 in &2 type *&3 not found.
CPF2113 E	Cannot allocate library &1.
CPF2173 E	Value for ASPDEV not valid with special value for library.
CPF218C E	&1 not a primary or secondary ASP.
CPF3141 E	Member &2 not found.
CPF34C0 E	Value &1 for number of fields to return parameter not valid.
CPF3C07 E	Error occurred while retrieving information from object &1.
CPF3C1D E	Length specified in parameter &1 not valid.
CPF3C21 E	Format name &1 is not valid.
CPF3C29 E	Object name &1 is not valid.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C39 E	Value for reserved field not valid.
CPF3C3B E	Value for parameter &2 for API &1 not valid.
CPF3C82 E	Key &1 not valid for API &2.
CPF3C89 E	Key &1 specified more than once.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF4268 E	Object &1 in &2 type *&3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9804 E	Object &2 in library &3 damaged.
CPF980B E	Object &1 in library &2 not available.
CPF9810 E	Library &1 not found.
CPF9812 E	File &1 in library &2 not found.
CPF9814 E	Device &1 not found.
CPF9820 E	Not authorized to use library &1.
CPF9822 E	Not authorized to file &1 in library &2.
CPF9825 E	Not authorized to device &1.
CPF9826 E	Cannot allocate file &2.
CPF9830 E	Cannot assign library &1
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB401 E	API &3 failed with reason code &1.
CPFB8ED E	Device description &1 not correct for operation.

Related Information

- the <**qdbst.h**> include file in library QSYSINC, for API related structure declarations and special value declarations
- the <**qdbstmgr.h**> include file in library QSYSINC, for the QdbstDeleteStatistics API prototype
- the <**qdbstds.h**> include file in library QSYSINC, for the QDBSTDS API prototype
- "Cancel Requested Statistics Collections (QDBSTCRS, QdbstCancelRequestedStatistics) API" on page 373 (QDBSTCRS, QdbstCancelRequestedStatistics) API
- "List Requested Statistics Collections (QDBSTLRS, QdbstListRequestedStatistics) API" on page 392 (QDBSTLRS, QdbstListRequestedStatistics) API
- "List Statistics Collection Details (QDBSTLDS, QdbstListDetailStatistics) API" on page 401 (QDBSTLDS, QdbstListDetailStatistics) API
- "List Statistics Collections (QDBSTLS, QdbstListStatistics) API" on page 412 (QDBSTLS, QdbstListStatistics) API
- "Request Statistics Collections (QDBSTRS, QdbstRequestStatistics) API" on page 426 (QDBSTRS, QdbstRequestStatistics) API

- "Update Statistics Collections (QDBSTUS, QdbstUpdateStatistics) API" on page 437 (QDBSTUS, QdbstUpdateStatistics) API

API introduced: V5R2

Top | "Database and File APIs," on page 1 | APIs by category

Dump SQL Database Monitor (QQQDSDBM) API

Required Parameter Group:

1
Memory handle to dump

Input
Char(10)

2
Job or memory handle name

Input
Char(26)

3
Number of types to dump

Input
Binary(4)

4
Subtypes and files array

Input
Array(*) of Char(30)

5
Error code

I/O
Char(*)

Service Program Name: QQQDSDBM

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 389.

The Dump SQL Database Monitor (QQQDSDBM) API dumps the SQL database monitor that has been gathered. The data that is gathered will be all data that has been committed (if the job is under commitment control) or based on a 5-minute timer. Associated APIs include the following:

- Clear SQL Database Monitor Statistics (QQQCSDBM)
- End SQL Database Monitor (QQQESDBM)
- Query SQL Database Monitor (QQQQSDBM)
- Start SQL Database Monitor (QQQSSDBM)

Authorities and Locks

Current User Profile
*JOBCTL

Library Authority for New File
: *ADD and *READ

Library Authority for Existing File
: *EXECUTE

Existing File
: *CHANGE and *OBJALTER

Required Parameter Group

Memory area to dump
INPUT; CHAR(10)

Memory area to dump. The possible values are:

**ALL*	Dump the monitor data associated with the *ALL monitor (the monitor started against all jobs). The Start SQL Database Monitor (QQQSSDBM) must have been started with job name *ALL.
**NAMED*	Dump the memory handle named by the job or memory handle name parameter (and matches the name of a memory handle specified on the QQQSSDBM API). Only the first 6 characters will be used for naming the memory handle. If QQQSSDBM started the monitor with *JOB, you can also name the job to be dumped with this parameter by giving the 6-character memory handle that contains the job number.
**JOB*	Dump the job-specific data associated with the job named by the job or memory handle name parameter.

Job or memory handle name
INPUT; CHAR(26)

This parameter depends on the value specified for the memory area to dump parameter. If the value is:

**ALL*	This parameter is ignored.
**NAMED*	The CHAR(6) name of a memory handle whose data is to be dumped. Only the first 6 characters will be used for naming the memory handle.
**JOB*	The CHAR(26) qualified job name of a job-specific monitor to dump. The qualified job name has three parts:

Job name
CHAR(10). A specific job name, a generic name, or one of following special values:

** or *CURRENT*
Only the job that this program is running in. The rest of the qualified job name parameter must be blank.

**ALL* All jobs. The rest of the job name parameter must be blank.

User name
CHAR(10). A specific user profile name.

Job number
CHAR(6). A specific job number.

Number of types to dump
INPUT; BINARY(4)

The number of types passed in the subtypes and files array.

Subtypes and files array
INPUT; Array(*) of CHAR(30)

The list of all subtypes to dump and their associated receiving files. The format of each array element is:

CHAR(10). Key to Dump.

The possible values are:

KEYT_3000	Summary: Arrival sequence
KEYT_3001	Summary: Index used
KEYT_3002	Summary: Index created
KEYT_3003	Summary: Sort
KEYT_3004	Summary: Temporary file
KEYT_3007	Summary: Optimizer time-out or all access paths considered
KEYT_3008	Summary: Subselect processing
KEYT_3010	Summary: Host variable values
KEYT_TEXT	SQL statement text
KEYT_QRYI	Summary: General SQL information including statement count, maximum runtime, time last used, and so forth. This subtype is always monitored because it is required for monitoring all other subtypes. Although it is always monitored, it will not be dumped unless requested.

CHAR(20). File name.

The name of the file to receive the data. The first 10 characters contain the file name, and the second 10 characters contain the library name. A member name of *FIRST is assumed. The following special values can be used for the library name:

**CURLIB*	The job's current library
**LIBL*	The library list

If the file already exists, it will be cleared prior to dumping the data. The files will be created with authority of owner(*ALL) and PUBLIC(*CHANGE), and they are owned by the profile of the job calling QQQDSDBM.

Note: If a subtype is specified multiple times, only the first occurrence of the subtype and the associated file name is honored. The duplicates are ignored.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

This function is threadsafe but not thread enabled. Database monitor data is collected in the threaded process but summarized at the job level.

The QQQDSDBM API does not force a clear operation (QQQCSDBM) of the memory. Data will continue to be added to memory until the QQQCSDBM or QQQESDBM API is called.

Error Messages

Message ID	Error Message Text
CPD0172 D	Parameters passed on CALL do not match those required.
CPD222E E	&1 special authority is required.

Message ID	Error Message Text
CPF3012 E	File &1 in library &2 not found.
CPF3084 E	Eror clearing member &3 in file &1.
CPF3130 E	Member &2 already in use.
CPF3CF1 E	Error code parameter not valid.
CPF9822 E	Not authorized to file &1 in library &2.

API introduced: V4R3

Top | "Database and File APIs," on page 1 | APIs by category

End SQL Database Monitor (QQQESDBM) API

Required Parameter Group:

1	Qualified job name	
Input	Char(26)	
2	Error code	
I/O	Char(*)	

Service Program Name: QQQESDBM

Default Public Authority: *USE

Threadsafe: Yes

The End SQL Database Monitor (QQQESDBM) API ends the memory-based SQL database monitor. Associated APIs include the following:

- Clear SQL Database Monitor Statistics (QQQCSDBM)
- Dump SQL Database Monitor (QQQDSDBM)
- Query SQL Database Monitor (QQQQSDBM)
- Start SQL Database Monitor (QQQSSDBM)

Authorities and Locks

Current User Profile
*JOBCTL

Required Parameter Group

Qualified job name
INPUT; CHAR(26)

The job to end monitoring on. The qualified job name has three parts:

Job name	CHAR(10). A specific job name, a generic name, or one of the following special values:
	** or *CURRENT* Only the job that this program is running in. The rest of the qualified job name parameter must be blank.
	**ALL* All jobs. The rest of the job name parameter must be blank.
User name	CHAR(10). A specific user profile name.
Job number	CHAR(6). A specific job number.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	**Error Message Text**
CPD0172 D	Parameters passed on CALL do not match those required.
CPF1321 E	Job &1 user &2 job number &3 not found.
CPF3CF1 E	Error code parameter not valid.
CPF436D E	Job specified is not being monitored.
CPF436E E	Job &1 user &2 job number &3 is not active.

API introduced: V4R3

Top | "Database and File APIs," on page 1 | APIs by category

List Requested Statistics Collections (QDBSTLRS, QdbstListRequestedStatistics) API

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format of output	Input	Char(8)
3	Input data	Input	Char(*)
4	Length of input data	Input	Binary(4)
5	Format of input data	Input	Char(8)
6	Error code	I/O	Char(*)

Service Program Name: QDBSTMGR

Default Public Authority: *USE

Threadsafe: Yes

The List Requested Statistics Collections (QDBSTLRS, QdbstListRequestedStatistics) API lists details for not yet completed, or not successfully completed statistics collection requests, requested by a call to the Request statistics collections (OPM, QDBSTRS; ILE QdbstRequestStatistics) API, the "Update Statistics Collections (QDBSTUS, QdbstUpdateStatistics) API" on page 437, or automatically by the system. The generated list replaces any existing list in the user space.

Section overview

Authorities and Locks

User Space Authority
: *CHANGE

User Space Library Authority
: *EXECUTE

User Space Lock
: *EXCLRD

ASP Device Authority
: *EXECUTE

File Authority
: *OBJOPR

File Library Authority
: *EXECUTE

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)

The user space that is to receive the generated list and the library in which it is located. The first 10 characters contain the user space name, and the second 10 characters contain the library name. You can use these special values for the library name:

**CURLIB*	The job's current library or QGPL if the current library is not set.
**LIBL*	The library list.
**USRLIBL*	Libraries listed in the user portion of the library list.

Format of output
INPUT; CHAR(8)

The format of the statistics collections list to be returned. If Format STOP0100 is specified, the fields that were selected by the caller are returned for each statistics collection request in the list. Possible format names are:

STOP0100	Statistics collection requests list with keyed return fields.

Refer to "Format of the Generated List" on page 395 and "List Data Section - STOP0100 Output Format" on page 396 for more information.

Input data
INPUT; CHAR(*)

The buffer containing the input parameters according to the format of input data parameter. The buffer content has to start at a 4-byte boundary.

Length of input data
INPUT; BINARY(4)

The length of the input data buffer provided.

Format of input data
INPUT; CHAR(8)

The format of the input data. Possible values are:

STIP0100	List Requested Statistics Collections input parameters.

Refer to “STIP0100 Format” for more information.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error code parameter.

STIP0100 Format

The following table shows the input parameters for this API. See “Field Descriptions” on page 396 for details of the fields listed.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	List filter option
4	4	CHAR(48)	Continuation handle
52	34	BINARY(4)	Offset to fields to return
56	38	BINARY(4)	Number of fields to return
60	3C	CHAR(*)	Reserved
		Array(*) of BINARY(4)	Keys of fields to return
		CHAR(*)	Reserved

Valid Keys - Request output

The keys listed below are used to determine in the “STIP0100 Format” what will be returned per list entry in the “List Data Section - STOP0100 Output Format” on page 396. Each key can only be specified once. See “Field Descriptions” on page 396 for details of the fields listed.

Each list entry returned in the output format describes a single request and can be thought of as two groups of related keys:

Group 1: The following keys describe information at request level and are repeated in list entries describing the single statistics collections within a request:

Key	Type	Description
1	CHAR(10)	ASP device name used
2	CHAR(10)	File name used
3	CHAR(10)	File library name used
4	CHAR(10)	File member name used
6	CHAR(16)	Internal request ID
34	CHAR(10)	Name of requesting user profile
35	CHAR(26)	Time stamp of request
36	CHAR(1)	Request status
37	CHAR(26)	Time stamp
38	CHAR(26)	Qualified job name
39	CHAR(8)	Thread ID
40	BINARY(4)	Progress percentage
8	BINARY(4)	Estimated time

Key	Type	Description
42	CHAR(7)	Message ID
49	CHAR(*)	Message Data
50	CHAR(10)	Message File Library
51	CHAR(10)	Message File
43	BINARY(4)	Total number of statistics collections for internal request ID

Group 2: The following keys describe information at statistics collection level for a request:

Key	Type	Description
44	BINARY(4)	Running number of statistics collection for internal request ID
46	CHAR(*)	Statistics collection name
18	CHAR(10)	Aging mode
28	BINARY(4)	Number of columns
29	Array(*) of CHAR(10)	Column names
30	Array(*) of CHAR(20)	Qualified translation table names

Format of the Generated List

The statistics collections list consists of:

- A user area
- A generic header
- “Input Parameter Section”
- “Header Section” on page 396
- “List Data Section - STOP0100 Output Format” on page 396

The user area and generic header are described in User space format for list APIs. The remaining items are described in the following sections.

Input Parameter Section

The following information is returned in the input parameter section. For detailed descriptions of the fields in this table, see “Field Descriptions” on page 396.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	User space name specified
10	A	CHAR(10)	User space library name specified
20	14	CHAR(8)	Format of output specified
28	1C	BINARY(4)	Length of input data specified
32	20	CHAR(8)	Format of input data specified
40	28	BINARY(4)	List filter option specified
44	2C	CHAR(48)	Continuation handle specified
92	5C	BINARY(4)	Offset to fields to return specified
96	60	BINARY(4)	Number of fields to return specified
100	64	BINARY(4)	Displacement to specified fields to return

Offset			
Dec	Hex	Type	Field
104	68	Array(*) of BINARY(4)	Keys of fields to return specified

Header Section

For detailed descriptions of the fields in this table, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	CHAR(48)	Continuation handle
48	30		

List Data Section - STOP0100 Output Format

The following information is returned in the list data section per statistics collection list entry for Format STOP0100. The fields are returned in the order requested. See "Field Descriptions" for details of the fields listed.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Length of list entry
4	4	BINARY(4)	Number of key fields returned
These fields repeat, in the order listed, for each key selected.		BINARY(4)	Length of field information returned
		BINARY(4)	Key identifier
		BINARY(4)	Length of data
		CHAR(*)	Data
		CHAR(*)	Reserved (padding to the next 4-byte boundary)

Field Descriptions

ASP device name used. The actual auxiliary storage pool device name used, after possible resolution of special values.

Note: If the API caller does not have the documented ASP Device Authority for the ASP device, the ASP device name returned will be set to all blanks.

Aging mode. Whether the system is allowed to age or remove the statistics collection. The possible values are:

**SYS*	Refresh or removal of the resulting statistics collections will be performed automatically by the system.
**USER*	Refresh or removal will only occur when a user requests it.

Column names. The array of names of the columns within the statistics collection, in the same order as at request time. Each array elements also corresponds to the array element at the same position in the Qualified translation table names field. The array dimension is given by the Number of columns field.

Note: If the API caller does not have the documented *ASP Device Authority* , *File Authority*, and *File Library Authority* for the file containing these specific columns, the column names returned will be set to all blanks.

Continuation handle (input section). The handle used to continue from a previous call to this API that resulted in partially complete information. You can determine if a previous call resulted in partially complete information by checking the Information Status variable in the generic user space header following the API call.

If the API is not attempting to continue from a previous call, this parameter must be set to blanks. Otherwise, a valid continuation value must be supplied. The value may be obtained from the list header section of the user space used in the previous call. When continuing, the first entry in the returned list is the entry that immediately follows the last entry returned in the previous call.

Continuation handle (header section). A continuation point for the API. This value is set based on the contents of the Information Status variable in the generic header for the user space. The following situations can occur:

Information status-C	The information returned in the user space is valid and complete. No continuation is necessary and the continuation handle is set to blanks.
Information status-P	The information returned in the user space is valid but incomplete. The user may call the API again, starting where the last call left off. The continuation handle contains a value which may be supplied as an input parameter in later calls.
Information status-I	The information returned in the user space is not valid and incomplete. The content of the continuation handle is unpredictable.

Data. The data returned for the key identifier.

Displacement to specified fields to return. The displacement to the start of the array of specified fields to return.

Note: This is not the offset specified on input, but the displacement within the input parameter section. See Offset to fields to return specified instead.

Estimated time. The estimated time in seconds to collect the statistics for this request. This will be zero for all request statuses but *'0' (pending)* and *'1' (active)*.

Note: For request status *'1' (active)*, the estimate is for the complete request, not for the remaining work to be done. The also returned progress percentage can be used to calculate the estimated time left before the request is complete.

File library name used. The actual file library name used, after possible resolution of special values.

Note: If the API caller does not have the documented *ASP Device Authority* and *File Library Authority* for this specific library, the library name returned will be set to all blanks.

File member name used. The actual file member name used, after possible resolution of special values.

Note: If the API caller does not have the documented ASP Device Authority, File Authority, and File Library Authority for the file containing this specific member, the file member name returned will be set to all blanks.

File name used. The actual file name used.

Note: If the API caller does not have the documented *ASP Device Authority*, *File Authority*, and *File Library Authority* for this specific file, the file name returned will be set to all blanks.

Internal request ID. Uniquely identifies a requested statistics collections. See the "Request Statistics Collections (QDBSTRS, QdbstRequestStatistics) API" on page 426.

Key identifier. The field returned. For a list of valid keys see "Valid Keys - Request output" on page 394.

Keys of fields to return. The list of fields to return per list entry. For a list of valid keys see "Valid Keys - Request output" on page 394.

Length of data. The length of the data returned for the field.

Length of field information returned. Total number of bytes returned for this field.

Length of list entry. Number of bytes returned for this list entry.

List filter option. The statistics collection requests to return. The filter option is a bit field and can be computed by adding up desired single filter values from the following list:

1	List requests in *'0' (pending)* status.
2	List requests in *'1' (active)* status.
4	List requests in *'2' (error)* status.

For information on the request status, see the Request Status (page 399) field.

Message ID. For request status *'2' (error)* only: A message ID describing the error.

Message File. For request status *'2' (error)* only: The message file for the *Message ID*.

Message File Library. For request status *'2' (error)* only: The library where the *Message File* is located.

Message Data. For request status *'2' (error)* only: The message field data for the *Message ID*.

Name of requesting user profile. The name of user profile that requested the statistics collection. The name will be *SYS for statistics collections automatically requested by the system.

Number of columns. Number of columns within the single statistics collection.

Number of fields to return. The number of fields to return for each list entry.

Number of key fields returned. Number of fields actually returned.

Offset to fields to return specified. Offset to fields to return as specified on the call of the API.

Progress percentage. For request status *'1' (active)* only: The percentage of completion of the request. For all other request statuses this will be zero.

Qualified job name. Depending on the request status, the following values are valid:

'0' (pending)	The job that submitted the request.
'1' (active)	The job processing the request.
'2' (error)	The job that did process the request.

The qualified job name has three parts:

Job name	Char(10).
User name	Char(10).

Job number	Char(6).

Qualified translation table names. The array of names of the translation tables that were specified, when the statistics collection was requested, in the same order as requested. Each array elements corresponds to the array element at the same position in the Column names field. The first 10 characters contain the translation table name, and the second 10 characters contain the name of the library where the table is located. The array dimension is given by the Number of columns field.

Note: For system initiated requests, the translation table name and the library can be set to the following special value:

**UNKNOWN*	The information is not available in a suitable form.

Request status. The current status of the requested statistics collection. The possible values are:

'0'	Pending. Request is scheduled for later processing.
'1'	Active. Request is currently being processed.
'2'	Error. Request processing did end in error and no statistics data was stored. **Note:** Only the most recently-ended requests are listed.

Reserved. Reserved for future use. If this field is input, the field must be set to hexadecimal zeros.

Reserved (in STOP0100 output format). Structure padding to guarantee alignment to the next 4-byte boundary.

Running number of statistics collections for internal request ID. Current index of the statistics collection definition for this request. The statistics collection definitions will be returned in the same order as requested.

Statistics collection name. A name unique among all statistics collections for the file member.

Thread ID. For request status *'1' (active)* only: The thread that is currently processing the request.

Time stamp. Depending on the request status, this time stamp shows:

'0' (pending)	The time the request was made (same as time stamp of request).
'1' (active)	The time processing started.
'2' (error)	The time the request ended in error.

Time stamp of request. The time stamp showing when the statistics collection was requested.

Total number of statistics collections for internal request ID. The number of statistics collection definitions for this request, identified by the internal request ID.

User space name specified. The user space name as specified on the call of the API.

User space library name specified. The user space library name as specified on the call of the API.

Error Messages

Message ID	Error Message Text
CPF0623 E	Field &1 not found in record format &2.
CPF1866 E	Value &1 for number of fields to return not valid.
CPF2105 E	Object &1 in &2 type *&3 not found.
CPF2113 E	Cannot allocate library &1.
CPF2173 E	Value for ASPDEV not valid with special value for library.
CPF218C E	&1 not a primary or secondary ASP.
CPF3141 E	Member &2 not found.
CPF34C0 E	Value &1 for number of fields to return parameter not valid.
CPF3C07 E	Error occurred while retrieving information from object &1.
CPF3C1D E	Length specified in parameter &1 not valid.
CPF3C21 E	Format name &1 is not valid.
CPF3C29 E	Object name &1 is not valid.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C39 E	Value for reserved field not valid.
CPF3C3B E	Value for parameter &2 for API &1 not valid.
CPF3C82 E	Key &1 not valid for API &2.
CPF3C89 E	Key &1 specified more than once.
CPF3CE2 E	Continuation handle not valid.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF4268 E	Object &1 in &2 type *&3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9804 E	Object &2 in library &3 damaged.
CPF980B E	Object &1 in library &2 not available.
CPF9810 E	Library &1 not found.
CPF9812 E	File &1 in library &2 not found.
CPF9814 E	Device &1 not found.
CPF9820 E	Not authorized to use library &1.
CPF9822 E	Not authorized to file &1 in library &2.
CPF9825 E	Not authorized to device &1.
CPF9826 E	Cannot allocate file &2.
CPF9830 E	Cannot assign library &1
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB401 E	API &3 failed with reason code &1.
CPFB8ED E	Device description &1 not correct for operation.

Related Information

- the <**qdbst.h**> include file in library QSYSINC, for API-related structure declarations and special value declarations.
- the <**qdbstmgr.h**> include file in library QSYSINC, for the QdbstListRequestedStatistics API prototype.
- the <**qdbstlrs.h**> include file in library QSYSINC, for the QDBSTLRS API prototype.
- "Cancel Requested Statistics Collections (QDBSTCRS, QdbstCancelRequestedStatistics) API" on page 373 (QDBSTCRS, QdbstCancelRequestedStatistics) API
- "Delete Statistics Collections (QDBSTDS, QdbstDeleteStatistics) API" on page 381 (QDBSTDS, QdbstDeleteStatistics) API
- "List Statistics Collection Details (QDBSTLDS, QdbstListDetailStatistics) API" on page 401 (QDBSTLDS, QdbstListDetailStatistics) API

- “List Statistics Collections (QDBSTLS, QdbstListStatistics) API” on page 412 (QDBSTLS, QdbstListStatistics) API
- “Request Statistics Collections (QDBSTRS, QdbstRequestStatistics) API” on page 426 (QDBSTRS, QdbstRequestStatistics) API
- “Update Statistics Collections (QDBSTUS, QdbstUpdateStatistics) API” on page 437 (QDBSTUS, QdbstUpdateStatistics) API

API introduced: V5R2

Top | “Database and File APIs,” on page 1 | APIs by category

List Statistics Collection Details (QDBSTLDS, QdbstListDetailStatistics) API

Required Parameter Group:

1	Qualified user space name
Input	Char(20)
2	Format of output
Input	Char(8)
3	Input data
Input	Char(*)
4	Length of input data
Input	Binary(4)
5	Format of input data
Input	Char(8)
6	Error code
I/O	Char(*)

Service Program Name: QDBSTMGR

Default Public Authority: *USE

Threadsafe: Yes

The List Statistics Collection Details (QDBSTLDS, QdbstListDetailStatistics) API lists additional statistics data for a single statistics collection, not returned by the “List Statistics Collections (QDBSTLS, QdbstListStatistics) API” on page 412. Available additional data is the list of most frequent values and the list of histogram ranges values.

The generated list output replaces any existing list in the user space.

Section overview

- “Authorities and Locks” on page 402
- “Required Parameter Group” on page 402
  - “STIV0100 Input Format” on page 403

Authorities and Locks

User Space Authority
*CHANGE

User Space Library Authority
*EXECUTE

User Space Lock
*EXCLRD

ASP Device Authority
*EXECUTE

File Authority
*OBJOPR, *READ

File Library Authority
*EXECUTE

File Lock
*SHRRD

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)

The user space that is to receive the generated list, and the library in which it is located. The first 10 characters contain the user space name, and the second 10 characters contain the library name. You can use these special values for the library name:

**CURLIB*	The job’s current library or QGPL if the current library is not set.
**LIBL*	The library list.
**USRLIBL*	Libraries listed in the user portion of the library list.

Format of output
INPUT; CHAR(8)

The format of the statistics collection details list to be returned. Possible format names are:

STOV0100	Statistics collection details list with list entries for the requested keys.

Refer to “Format of the Generated List” on page 404 and “List Data Section - STOV0100 Format” on page 406 for more information.

Input data
INPUT; CHAR(*)

The buffer containing the input parameters according to the format of input data parameter. The buffer content has to start at a 4-byte boundary.

Length of input data
INPUT; BINARY(4)

The length of the input data buffer provided.

Format of input data
INPUT; CHAR(8)

The format of the input data. Possible values are:

STIV0100	List statistics collection details input parameters.

Refer to “STIV0100 Input Format” for more information.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

STIV0100 Input Format

See “Field Descriptions” on page 406 for details of the fields listed.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	ASP device name
10	0A	CHAR(10)	File name
20	14	CHAR(10)	File library name
30	1E	CHAR(10)	File member name
40	28	CHAR(16)	Internal statistics ID
56	38	CHAR(48)	Continuation handle
104	68	BINARY(4)	Offset to fields to return
108	6C	BINARY(4)	Number of fields to return
112	70	CHAR(*)	Reserved
		Array(*) of BINARY(4)	Keys of fields to return
		CHAR(*)	Reserved

Valid Keys - Request output

Each key can only be specified once. See “Field Descriptions” on page 406 for details of the fields listed.

Key	Type	Description
32	CHAR(*)	“Most Frequent Values List Entry Format” on page 406
33	CHAR(*)	“Histogram Range Values List Entry Format” on page 406

Format of the Generated List

The statistics collection details list consists of:

- A user area
- A generic header
- "Input Parameter Section"
- A header section
- "List Data Section - STOV0100 Format" on page 406

The user area and generic header are described in User space format for list APIs. The remaining items are described in the following sections.

Input Parameter Section

The following information is returned in the input parameter section. For detailed descriptions of the fields in this table, see "Field Descriptions" on page 406.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	User space name specified
10	A	CHAR(10)	User space library name specified
20	14	CHAR(8)	Format of output specified
28	1C	BINARY(4)	Length of input data specified
32	20	CHAR(8)	Format of input data specified
40	28	CHAR(10)	ASP device name specified
50	32	CHAR(10)	File name specified
60	3C	CHAR(10)	File library name specified
70	46	CHAR(10)	File member name specified
80	50	CHAR(48)	Continuation handle specified
128	80	BINARY(4)	Offset to fields to return specified
132	84	BINARY(4)	Number of fields to return specified
136	88	BINARY(4)	Displacement to specified fields to return
		Array(*) of BINARY(4)	Keys of fields to return specified

Header Section

For detailed descriptions of the fields in this table, see "Field Descriptions" on page 406.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	ASP device name used
10	A	CHAR(10)	File library name used
20	14	CHAR(10)	File member name used
30	1E	CHAR(2)	Reserved
32	20	CHAR(48)	Continuation handle

Offset			
Dec	Hex	Type	Field
80	50	BINARY(4)	Displacement to detail values header of most frequent values
84	54	BINARY(4)	Displacement to detail values header of histogram range values
88	58		

Detail Values Header

This structure contains fields describing general information for list entries of a certain kind (most frequent values or histogram range values) returned in the list data section.

See "Field Descriptions" on page 406 for details of the fields listed.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Offset to first detail value entry
4	4	BINARY(4)	Number of detail value entries
8	8	BINARY(4)	Length of detail value entry
12	C	BINARY(4)	Number of detail value columns
16	10	BINARY(4)	Displacement to "Detail Value Format Description" of first detail value column.
20	14	BINARY(4)	Length of detail value column format
24	18		

Detail Value Format Description

The description of a single detail value column value. See "Field Descriptions" on page 406 for details of the fields listed.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	SQL data type
4	4	BINARY(4)	Field length
8	8	BINARY(4)	Length in bytes
12	C	BINARY(4)	Scale
16	10	BINARY(4)	Precision
20	14	BINARY(4)	Radix
24	18	BINARY(4)	CCSID
28	1C	CHAR(10)	Translation table name
38	26	CHAR(10)	Translation table library name
48	30	CHAR(1)	DDS type
49	31	CHAR(3)	Reserved
52	34		

List Data Section - STOV0100 Format

The list data section returned for output format STOV0100 contains list entries as specified in the "Most Frequent Values List Entry Format" and the "Histogram Range Values List Entry Format."

See the Header Section for additional fields that describe information common to all Most frequent value and Histogram range value list entries.

Most Frequent Values List Entry Format

The format below describes the layout of a single most frequent value returned as a list entry in the list data section. See "Field Descriptions" and the Header Section for details of the fields listed.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(8)	Count for this most frequent value
0	0	BINARY(4)	Displacement to first most frequent value column value
0	0	CHAR(4)	Reserved
		Array of CHAR(*)	Most frequent value columns values
		CHAR(*)	Reserved

Histogram Range Values List Entry Format

This format describes the layout of a single histogram range value returned as a list entry in the list data section. See "Field Descriptions" and the Header Section for details of the fields listed.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(8)	Count for this histogram range value
0	0	BINARY(8)	Count for this histogram range's high value
0	0	BINARY(4)	Displacement to first histogram range value column pair
0	0	CHAR(4)	Reserved
		Array of CHAR(*)	Low/High value pairs of histogram range value columns
		CHAR(*)	Reserved

Field Descriptions

ASP device name. The name of one auxiliary storage pool (ASP) device in the ASP group in which the library and file are located. The ASP device must have a status of 'Available'. The documented authority is required for the given ASP and the primary of the corresponding ASP group. The name can be a specific ASP device name (for an ASP with a number greater than 32), or one of the following special values:

*	Locate the library and file in the name space for the current thread.
**SYSBAS*	Locate the library and file in the system ASP (ASP number 1) and all basic ASPs (ASP numbers 2 through 32).

ASP device name used. The actual auxiliary storage pool device name used, after possible resolution of special values.

CCSID. The value's CCSID for character type values.

Note: The CCSID here describes the CCSID of the original value, before translation using the also given translation table (page 410).

Continuation handle (input section). The handle used to continue from a previous call to this API that resulted in partially complete information. You can determine if a previous call resulted in partially complete information by checking the Information Status variable in the generic user space header following the API call.

If the API is not attempting to continue from a previous call, this parameter must be set to blanks. Otherwise, a valid continuation value must be supplied. The value may be obtained from the list header section of the user space used in the previous call. When continuing, the first entry in the returned list is the entry that immediately follows the last entry returned in the previous call.

Continuation handle (header section). A continuation point for the API. This value is set based on the contents of the Information Status variable in the generic header for the user space. The following situations can occur:

Information status-C	The information returned in the user space is valid and complete. No continuation is necessary and the continuation handle is set to blanks.
Information status-P	The information returned in the user space is valid but incomplete. The user may call the API again, starting where the last call left off. The continuation handle contains a value which may be supplied as an input parameter in later calls.
Information status-I	The information returned in the user space is not valid and incomplete. The content of the continuation handle is unpredictable.

Count for this histogram range's high value. If the statistics collection key chosen for the high value of this histogram range value is estimated to occur very often, compared to the other values in this histogram range, this count will be set to the estimated number of occurrences of this key value in the file member.

Note: A value of 0 indicates, that this additional information about the high value is not available.

Count for this histogram range value. How many statistics collection key values are estimated to occur in this histogram range.

Count for this most frequent value. How often the most frequent statistics collection key value is estimated to occur in the file member.

DDS type. Data type code (corresponding to the *SQL Data Type*) for the value. See DDS Reference: Physical and Logical Files.

Displacement detail values header of histogram range values. Displacement to the general information for the histogram range values list entries.

Note: The displacement will be zero if no histogram range value information was returned.

Displacement to detail values header of most frequent values. Displacement to the general information for the most frequent values list entries. The displacement will be zero if no most frequent value information was returned.

Displacement to first histogram range value column pair. Displacement to the start of the array of *Low/High value pairs of histogram range value columns*.

Displacement to first most frequent value column value. Displacement to the start of the array of *Most frequent value column values*.

Displacement to format of first detail value column. Displacement to the array of format descriptions for the detail value column values (compare *Number of detail value columns* and *Length of detail value column format*).

Note: Due to performance reasons, these formats might be different from the formats of the original columns the detail value is based on. For example, long character columns might be represented in a truncated form, or varying character columns might have been converted to a fixed length form.

Note: For DATE, TIME, and TIMESTAMP columns, the detail value column values will be returned as *ISO formatted text and the column format will describe a character SQL data type of appropriate length.

Displacement to specified fields to return. Displacement to the start of the array of specified fields to return.

Note: This is not the offset specified on input, but the displacement within the input parameter section. See the Offset to fields to return specified instead.

Field length. Value field length.

File library name. The location of the file for which statistics collection details are to be listed. You can use these special values for the library name, if the *ASP Device Name* is *:

**CURLIB*	The job's current library or QGPL if the current library is not set.
**LIBL*	The library list.
**USRLIBL*	Libraries listed in the user portion of the library list.

File library name used. The actual file library name used, after possible resolution of special values.

File member name. The name of the file member to be used for the list request. This value can be a specific file member name or one of the following special values:

**FIRST*	The first member (in the order created) in the specified file.
**LAST*	The last member (in the order created) in the specified file.

File member name used. The actual file member name used, after possible resolution of special values.

File name. The name of the file for which statistics collection details are to be listed. The file has to be an existing local, single format, physical file.

Histogram range values. The list of histogram range values. See "Histogram Range Values List Entry Format" on page 406 for the layout of this list.

Internal statistics ID. Together with the qualified file name and member name, this represents a unique ID for the statistics collection details to be listed.

Key identifier. The field returned. For a list of valid keys see "Valid Keys - Request output" on page 403.

Keys of fields to return. The list of fields in the list. For a list of valid keys see "Valid Keys - Request output" on page 403.

Length in bytes. Length of returned column value in list entry, in bytes. This also gives the displacement to the next column value, where appropriate.

Length of detail value column format. The number of bytes for a single detail value column format. This is also the offset to the next detail value column format, for any but the last detail value column format of this kind of detail values. (See *Number of detail value columns*).

Length of detail value entry. The number of bytes for a single detail value list entry. This is also the offset to the next detail value list entry for any but the last detail value list entry of this kind of detail values in the list data section (See *Number of detail value entries*).

Low/High value pairs of histogram range value columns. The array of lower, exclusive, and upper, inclusive, histogram range column values for each column in the statistics collection key. The values are returned in the following order: Low value of first range value column, high value of first range value column, ... , low value of last range value column, high value of last range value column. The array dimension is given by the *Number of detail value columns* and the formats of the column values are referenced by the *Displacement to format of first detail value column*. Lower and upper value column always have the same format for a single column in the statistics collection key. The size of a single value column is given by the *Length in bytes* in the format.

Note: The column values for the low value of the first histogram range will be set to all X'00' and should be treated as undefined, representing 'negative infinity'.

Most frequent value columns values. The array of this most frequent value's column values. The array dimension is given by the *Number of detail value columns* and the formats of the column values are referenced by the *Displacement to format of first detail value column*. The size of a single value column is given by the *Length in bytes* in the corresponding format.

Most frequent values. The list of most frequent values. See "Most Frequent Values List Entry Format" on page 406 for the layout of this list.

Number of detail value columns. Number of columns in the statistics key for this kind of detail values.

Number of detail value entries. Number of list entries for this kind of detail values (most frequent values or histogram range values) returned on this API call. This value will be zero, if this kind of detail value was not requested to be returned, or, if during this API call, no space was available anymore to fit any value information of this kind into the user space besides other requested detail information. Compare the *Continuation handle* in the Header Section, how to retrieve the remaining API information in such a case.

Number of fields to return. The number of fields to return in the list.

Offset to fields to return. Offset to the start of the array of fields to return.

Offset to fields to return specified. The offset to fields to return as specified on the call of the API.

Offset to first detail value entry. The offset to the start of the list of this kind of detail value entries within the "List Data Section - STOV0100 Format" on page 406.

Note: The offset is relative to the start of the user space. The offset is valid only if the Number of detail value entries field is not set to zero.

Offset to value of most frequent value column. The offset to the value of the most frequent value column.

Precision. The precision of the value for numeric data type values.

Radix. Whether the value precision is specified in number of binary or decimal digits for numeric data type values. The possible values are:

2	Value precision is number of binary digits.
10	Value precision is number of decimal digits.

Reserved. Reserved for future use. If this field is input, the field must be set to hexadecimal zeros.

SQL data type. The SQLTYPE of the value as explained in the SQL Reference.

Scale. The scale of the value for numeric data type values.

Translation table library name. The library where the translation table used was located. If no translation table was used, the library name is set to all blanks.

Translation table name. The translation table used on the value, when the statistics collection was created. If no translation table was used, the table name is set to all blanks.

Note: The value is actually returned in the translated form.

User space name specified. The user space name as specified on the call of the API.

User space library name specified. The user space library name as specified on the call of the API.

Error Messages

Message ID	Error Message Text
CPF0623 E	Field &1 not found in record format &2.
CPF1866 E	Value &1 for number of fields to return not valid.
CPF2105 E	Object &1 in &2 type *&3 not found.
CPF2113 E	Cannot allocate library &1.
CPF2173 E	Value for ASPDEV not valid with special value for library.
CPF218C E	&1 not a primary or secondary ASP.
CPF3141 E	Member &2 not found.
CPF34C0 E	Value &1 for number of fields to return parameter not valid.
CPF3C07 E	Error occurred while retrieving information from object &1.
CPF3C1D E	Length specified in parameter &1 not valid.
CPF3C21 E	Format name &1 is not valid.
CPF3C29 E	Object name &1 is not valid.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C39 E	Value for reserved field not valid.
CPF3C3B E	Value for parameter &2 for API &1 not valid.
CPF3C82 E	Key &1 not valid for API &2.
CPF3C89 E	Key &1 specified more than once.
CPF3CE2 E	Continuation handle not valid.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF4268 E	Object &1 in &2 type *&3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9804 E	Object &2 in library &3 damaged.
CPF980B E	Object &1 in library &2 not available.
CPF9810 E	Library &1 not found.
CPF9812 E	File &1 in library &2 not found.
CPF9814 E	Device &1 not found.

Message ID	Error Message Text
CPF9820 E	Not authorized to use library &1.
CPF9822 E	Not authorized to file &1 in library &2.
CPF9825 E	Not authorized to device &1.
CPF9826 E	Cannot allocate file &2.
CPF9830 E	Cannot assign library &1
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB401 E	API &3 failed with reason code &1.
CPFB8ED E	Device description &1 not correct for operation.

Related Information

- the <**qdbst.h**> include file in library QSYSINC, for API-related structure declarations and special value declarations.
- the <**qdbstmgr.h**> include file in library QSYSINC, for the QdbstListDetailStatistics API prototype.
- the <**qdbstlds.h**> include file in library QSYSINC, for the QDBSTLDS API prototype.
- "Cancel Requested Statistics Collections (QDBSTCRS, QdbstCancelRequestedStatistics) API" on page 373 (QDBSTCRS, QdbstCancelRequestedStatistics) API
- "Delete Statistics Collections (QDBSTDS, QdbstDeleteStatistics) API" on page 381 (QDBSTDS, QdbstDeleteStatistics) API
- "List Requested Statistics Collections (QDBSTLRS, QdbstListRequestedStatistics) API" on page 392 (QDBSTLRS, QdbstListRequestedStatistics) API
- "List Statistics Collections (QDBSTLS, QdbstListStatistics) API" on page 412 (QDBSTLS, QdbstListStatistics) API
- "Request Statistics Collections (QDBSTRS, QdbstRequestStatistics) API" on page 426 (QDBSTRS, QdbstRequestStatistics) API
- "Update Statistics Collections (QDBSTUS, QdbstUpdateStatistics) API" on page 437 (QDBSTUS, QdbstUpdateStatistics) API

API introduced: V5R2

Top | "Database and File APIs," on page 1 | APIs by category

List Statistics Collections (QDBSTLS, QdbstListStatistics) API

Required Parameter Group:

1	Qualified user space name	
Input	Char(20)	
2	Format of output	
Input	Char(8)	
3	Input data	
Input	Char(*)	
4	Length of input data	
Input	Binary(4)	
5	Format of input data	
Input	Char(8)	
6	Error code	
I/O	Char(*)	

Service Program Name: QDBSTMGR

Default Public Authority: *USE

Threadsafe: Yes

The List Statistics Collections (QDBSTLS, QdbstListStatistics) API allows to find out all of the columns and combination of columns for a given file member, which have statistics available and will optionally list those columns, not contained in any statistics collection. The generated list replaces any existing list in the user space.

Each returned list entry contains a number of different statistic data items, including the number of histogram ranges and the number of most frequent values, while detailed information for these two items can be retrieved using the "List Statistics Collection Details (QDBSTLDS, QdbstListDetailStatistics) API" on page 401 (QDBSTLDS, QdbstListDetailStatistics) API, using the internal statistics ID returned by the QdbstListStatistics API.

The QdbstListStatistics API also allows to list statistics related attributes at the file member level, not related to a single statistics collection.

Section overview

- "Authorities and Locks" on page 413
- "Required Parameter Group" on page 413
  - "STIL0100 Input Format" on page 414
  - "Valid Keys - Request output" on page 414
- "Format of the Generated List" on page 415
  - "Input Parameter Section" on page 416
  - "Header Section" on page 416
  - "List Data Section - STOL0100 Output Format" on page 416

Authorities and Locks

User Space Authority
*CHANGE

User Space Library Authority
*EXECUTE

User Space Lock
*EXCLRD

ASP Device Authority
*EXECUTE

File Authority
*OBJOPR

File Library Authority
*EXECUTE

File Lock
*SHRRD

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)

The user space that is to receive the generated list, and the library in which it is located. The first 10 characters contain the user space name, and the second 10 characters contain the library name.

You can use these special values for the library name:

**CURLIB*	The job's current library or QGPL if the current library is not set.
**LIBL*	The library list.
**USRLIBL*	Libraries listed in the user portion of the library list.

Format of output
INPUT; CHAR(8)

The format of the statistics collections list to be returned. If format **STOL0100** is specified, the fields that were selected by the caller will be returned for each statistics collection in the list. Possible format names are:

STOL0100	Statistics collections list with keyed return fields.

Refer to "Format of the Generated List" on page 415 and "List Data Section - STOL0100 Output Format" on page 416 for more information.

Input data
INPUT; CHAR(*)

The buffer containing the input parameters according to the format of input data parameter. The buffer content has to start at a four-byte boundary.

Length of input data
INPUT; BINARY(4)

The length of the input data buffer provided.

Format of input data
INPUT; CHAR(8)

The format of the input data. Possible values are:

STIL0100	List statistics collections input parameters.

Refer to "STIL0100 Input Format" for more information.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

STIL0100 Input Format

List statistics collections input parameters. See "Field Descriptions" on page 417 for details of the fields listed.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	ASP device name
10	A	CHAR(10)	File name
20	14	CHAR(10)	File library name
30	1E	CHAR(10)	File member name
40	28	CHAR(1)	Column option
41	29	CHAR(3)	Reserved
44	2C	CHAR(48)	Continuation handle
92	5C	BINARY(4)	Offset to fields to return
96	60	BINARY(4)	Number of fields to return
100	64	CHAR(*)	Reserved
		Array(*) of BINARY(4)	Keys of fields to return
		CHAR(*)	Reserved

Valid Keys - Request output

The keys listed below are used to determine in the "STIL0100 Input Format" what will be returned per list entry in the "List Data Section - STOL0100 Output Format" on page 416. Each key can only be specified once. See "Field Descriptions" on page 417 for details of the fields listed.

Each list entry returned in the output format describes a single statistics collection for a specific file member and can be thought of as two groups of related keys:

Group 1: The following keys describe information at file member level and will repeat in list entries describing different statistics collections (see group 2) for the same file member:

Key	Type	Description
1	CHAR(10)	ASP device name used
2	CHAR(10)	File name used
3	CHAR(10)	File library name used
4	CHAR(10)	File member name used
9	CHAR(26)	Current time stamp of last change
10	BINARY(8)	Current number of (undeleted) records.
11	BINARY(8)	Current number of deleted records.
12	BINARY(8)	Current total count of inserts, updates, and deletes.
47	CHAR(1)	Current block system statistics collections option.
48	BINARY(8)	Current size of statistics collections.

Group 2: The following keys describe information at statistics collection level per file member:

Key	Type	Description
7	CHAR(16)	Internal statistics ID
46	CHAR(*)	Statistics collection name
14	CHAR(10)	Name of creating user profile
15	CHAR(26)	Time stamp of create
52	CHAR(10)	Name of last modifying user profile
53	CHAR(26)	Time stamp of last modification
16	BINARY(4)	Number of most frequent values available
17	BINARY(4)	Number of histogram ranges available
18	CHAR(10)	Aging mode
19	CHAR(1)	Aging status
22	CHAR(1)	Translation attribute
23	BINARY(8)	Number of (undeleted) records
24	BINARY(8)	Number of deleted records
25	BINARY(8)	Total counts of inserts, updates, and deletes
26	BINARY(8)	Number of distinct values (cardinality)
27	BINARY(8)	Number of NULLs
28	BINARY(4)	Number of columns
29	Array(*) of CHAR(10)	Column names
41	Array(*) of CHAR(1)	Translation attributes
30	Array(*) of CHAR(20)	Qualified translation table names
31	Array(*) of CHAR(*)	Column descriptions (page 418)

Format of the Generated List

The statistics collections list consists of:

- A user area
- A generic header
- "Input Parameter Section" on page 416

- “Header Section”
- “List Data Section - STOL0100 Output Format”

The user area and generic header are described in User Space Format for List APIs. The remaining items are described in the following sections.

Input Parameter Section

The following information is returned in the input parameter section. For detailed descriptions of the fields in this table, see “Field Descriptions” on page 417.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	User space name specified
10	A	CHAR(10)	User space library name specified
20	14	CHAR(8)	Format of output specified
28	1C	BINARY(4)	Length of input data specified
32	20	CHAR(8)	Format of input data specified
40	28	CHAR(10)	ASP device name specified
50	32	CHAR(10)	File name specified
60	3C	CHAR(10)	File library name specified
70	46	CHAR(10)	File member name specified
80	50	CHAR(1)	Column option specified
81	51	CHAR(3)	Reserved
84	54	CHAR(48)	Continuation handle specified
132	84	BINARY(4)	Offset to fields to return specified
136	88	BINARY(4)	Number of fields to return specified
140	8C	BINARY(4)	Displacement to specified fields to return
		Array(*) of BINARY(4)	Keys of fields to return specified

Header Section

For detailed descriptions of the fields in this table, see “Field Descriptions” on page 417.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(48)	Continuation handle
48	30		

List Data Section - STOL0100 Output Format

For output format STOL0100, the list data section has the following layout, where each list entry contains the requested fields for a single statistics collection for a specific file member. See also “Valid Keys - Request output” on page 414 and note, that the fields for each list entry will be returned in the order requested.

See “Field Descriptions” on page 417 for details of the fields listed in the layout.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Length of list entry
4	4	BINARY(4)	Number of key fields returned
These fields repeat, in the order listed, for each key selected.		BINARY(4)	Length of field information returned
		BINARY(4)	Key identifier
		BINARY(4)	Length of data
		CHAR(*)	Data
		CHAR(*)	Reserved (padding to the next four-byte boundary)

Column Description

Layout of a single returned column description, if *Column descriptions* were requested as output in the "STIL0100 Input Format" on page 414. See "Field Descriptions" for details of the fields listed.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	SQL data type
4	4	BINARY(4)	Field length
8	8	BINARY(4)	Length in bytes
12	C	BINARY(4)	Scale
16	10	BINARY(4)	Precision
20	14	BINARY(4)	Radix
24	18	BINARY(4)	CCSID
28	1C	CHAR(1)	NULL capable
29	1D	CHAR(1)	Has default
30	1E	CHAR(50)	Column text
80	50	BINARY(4)	Ordinal position
84	54		

Field Descriptions

Aging mode. Whether the system is allowed to age or remove the statistics collection. The possible values are:

**SYS*	Refresh or removal of the resulting statistics collections will be performed automatically by the statistics manager.
**USER*	Refresh or removal will only occur when a user requests it.

Aging status. How current the statistics data is. The possible values are:

'0'	There are no indications, that the statistics data needs to be refreshed.
'1'	There are indications, that the statistics data needs to be refreshed.

ASP device name. The name of one auxiliary storage pool (ASP) device in the ASP group in which the library and file are located. The ASP device must have a status of 'Available'. The documented authority is required for the given ASP and the primary of the corresponding ASP group. The name can be a specific ASP device name (for an ASP with a number greater than 32), or one of the following special values:

*	Locate the library and file in the name space for the current thread.
**SYSBAS*	Locate the library and file in the system ASP (ASP number 1) and all basic ASPs (ASP numbers 2 through 32).

ASP device name used. The actual auxiliary storage pool device name used, after possible resolution of special values.

CCSID. The column CCSID for character type columns.

Column descriptions. The array of detailed column descriptions in the same order as the columns were requested. The array dimension is given by the *Number of Columns* field. See "Column Description" on page 417 for the layout of a single column description.

Column names. The array of names of the columns within the statistics collection, in the same order as at request time. The array dimension is given by the *Number of Columns* field.

Column option. Which columns and combination of columns to include in the list. The possible values are:

'0'	Do not include pseudo, single column statistics collection list entries for columns not contained in any actual statistics collection.
'1'	Do include pseudo, single column statistics collection list entries for columns not contained in any actual statistics collection. **Note:** Pseudo statistics collections will be marked by having an internal statistics ID of zero. All other statistics collection related fields for such a statistics collection list entry will be undefined, if requested to be returned.

Column text. The character string supplied with the LABEL ON SQL statement for this column.

Continuation handle (input section). The handle used to continue from a previous call to this API that resulted in partially complete information. You can determine if a previous call resulted in partially complete information by checking the Information Status variable in the generic user space header following the API call.

If the API is not attempting to continue from a previous call, this parameter must be set to blanks. Otherwise, a valid continuation value must be supplied. The value may be obtained from the list header section of the user space used in the previous call. When continuing, the first entry in the returned list is the entry that immediately follows the last entry returned in the previous call.

Continuation handle (header section). A continuation point for the API. This value is set based on the contents of the Information Status variable in the generic header for the user space. The following situations can occur:

Information status-C	The information returned in the user space is valid and complete. No continuation is necessary and the continuation handle is set to blanks.
Information status-P	The information returned in the user space is valid but incomplete. The user may call the API again, starting where the last call left off. The continuation handle contains a value which may be supplied as an input parameter in later calls.
Information status-I	The information returned in the user space is not valid and incomplete. The content of the continuation handle is unpredictable.

Current block system statistics collections option. Whether system initiated (automatic) statistics collection create requests are allowed for this database file member. The possible values are:

'0'	System initiated statistics collection requests are not blocked. **Note:** This is the system default.
'1'	System initiated statistics collection requests are blocked.

Current number of deleted records. The total count of deleted records in the file member at the time of the list request.

Current number of (undeleted) records. The total count of active records in the file member at the time of the list request.

Current size of statistics collections. The total amount of space in bytes used for statistics collections related data for this file member.

Current time stamp of last change. The time stamp, when the file member was last changed at the time of the list request.

Current total count of inserts, updates, and deletes. The number of insert, update, and delete operations that were recorded for the file member at the time of the list request.

Data. The data returned for the key identifier.

Displacement to specified fields to return. Displacement to the start of the array of specified fields to return.

Note: This is not the offset specified on input, but the displacement within the input parameter section. See the Offset to fields to return specified instead.

Field length. Column field length.

File library name. Where the file for which statistics collections are to be listed is located. You can use these special values for the library name, if the *ASP Device Name* is *:

**CURLIB*	The job's current library or QGPL if the current library is not set.
**LIBL*	The library list.
**USRLIBL*	Libraries listed in the user portion of the library list.

File library name used. The actual file library name used, after possible resolution of special values.

File member name. The name of the file member to be used for the list request. This value can be a specific file member name or one of the following special values:

**FIRST*	The first member (in the order created) in the specified file.
**LAST*	The last member (in the order created) in the specified file.
**ALL*	All members in the specified file.

File member name used. The actual file member name used, after possible resolution of special values.

File name. The name of the file for which statistics collections are to be listed. This can be a name of an existing local, single format, physical file. If an actual name is specified for the file library name, then you can also use the special value:

**ALL*	All local, single format, physical files in the specified library.

File name used. The actual file name used.

Has default. Whether the column has a default value (DEFAULT clause or null capable). The possible values are:

'0'	Column does not have a default value.
'1'	Column has default value.

Internal statistics ID. Together with the qualified file name and member name this represents a unique ID for the statistics collection listed.

Note: The ID is stored in binary, non printable form in the character array.

Key identifier. The field returned. For a list of valid keys see "Valid Keys - Request output" on page 414.

Keys of fields to return. The list of fields to return per list entry. For a list of valid keys see "Valid Keys - Request output" on page 414.

Length in bytes. Column length in bytes.

Length of data. The length of the data returned for the field.

Length of field information returned. Total number of bytes returned for this field.

Length of list entry. Number of bytes returned for this list entry.

Name of creating user profile. The name of the user profile, which requested the statistics collection. The name will be *SYS for statistics collections automatically requested by the system.

Name of last modifying user profile. The name of the user profile, which updated the statistics collection data last. The name will be *SYS for statistics collections automatically refreshed by the system.

Note: Updates of statistics collection attributes will not be logged here.

NULL capable. whether the column allows NULL values or not. The possible values are:

'0'	Column does not allow NULL values.
'1'	Column does allow NULL values.

Number of columns. Number of columns within the single statistics collection.

Number of deleted records. The total count of deleted records in the file member at the time the statistics were collected.

Number of distinct values. The estimated number of distinct (non NULL) values found in the statistics collection key.

Number of fields to return. The number of fields to return for each list entry.

Number of histogram ranges available. The number of histogram ranges available for this statistics collection. The actual histogram range values can be obtained using the "List Statistics Collection Details (QDBSTLDS, QdbstListDetailStatistics) API" on page 401 (QDBSTLDS, QdbstListDetailStatistics) API.

Number of key fields returned. Number of fields actually returned.

Number of most frequent values available. The number of most frequent values available for this statistics collection. The actual most frequent values can be obtained using the "List Statistics Collection Details (QDBSTLDS, QdbstListDetailStatistics) API" on page 401 (QDBSTLDS, QdbstListDetailStatistics) API.

Number of NULLs. The estimated number of NULL values found in the statistics collection key.

Number of (undeleted) records. The total count of active records in the file member at the time the statistics were collected.

Number of deleted records. The total count of deleted records in the file at the time the statistics were collected.

Offset to fields to return. Offset to the start of the array of fields to return.

Offset to fields to return specified. Offset to fields to return as specified on the call of the API.

Displacement to specified fields to return. Displacement to the start of the array of specified fields to return.

Note: This is not the offset specified on input, but the displacement within the input parameter section. See the Offset to fields to return specified instead.

Ordinal position. Numeric place of the column in the file member, ordered from left to right, starting with one.

Precision. The precision of the column for numeric data type columns.

Qualified translation table names. The array of names of the translation tables that were specified, when the statistics collection was requested, in the same order as requested. The first 10 characters contain the translation table name, and the second 10 characters contain the name of the library where the table is located. The array dimension is given by the number of columns field.

Note: For system initiated requests, the translation table name and the library can be set to the special value:

**UNKNOWN*	The information is not available in a suitable form.

Radix. whether the column precision is specified in number of binary or decimal digits for numeric data types columns. The possible values are:

2	Column precision is number of binary digits.
10	Column precision is number of decimal digits.

Reserved. Reserved for future use. If this field is input, the field must be set to hexadecimal zeros.

Reserved (in STOL0100 Output format). Structure padding to guarantee alignment to the next four bytes boundary.

Scale. The scale of the column for numeric data type columns.

SQL data type. The SQLTYPE of the column as explained in the SQL Reference.

Statistics collection name. A name unique amongst all statistics collections for the file member.

Time stamp of create. The time stamp, when the statistics collection was created.

Time stamp of last modification. The time stamp, when the statistics collection was last modified. This includes the initial create and any update of the statistics collection data.

Note: Updates to just statistics collection attributes will not be logged here.

Total count of inserts, updates, and deletes. The number of insert, update, and delete operations that were recorded for the file member at the time the statistics were collected.

Translation attribute. Indicates the type of translation used on the combination of character columns in the statistics collection key before the statistics were calculated. This attribute generalizes the information given by the the single translation attribute values returned for each column. The possible values are:

'0'	Uniquely weighted translation.
'1'	Shared weight translation.
'9'	No translation.

Translation attributes. The array of translation attributes for the single columns in the statistics collection key in the same order as requested. The translation attribute indicates the type of translation used on a character column before the statistics were calculated and generalizes the type of translation defined by the translation table applied to this column. The possible values for each array entry are:

'0'	Uniquely weighted translation.
'1'	Shared weight translation.
'9'	No translation.

User space name specified. User space name as specified on the call of the API.

User space library name specified. User space library name as specified on the call of the API.

Error Messages

Message ID	**Error Message Text**
CPF0623 E	Field &1 not found in record format &2.
CPF1866 E	Value &1 for number of fields to return not valid.
CPF2105 E	Object &1 in &2 type *&3 not found.
CPF2113 E	Cannot allocate library &1.
CPF2173 E	Value for ASPDEV not valid with special value for library.
CPF218C E	&1 not a primary or secondary ASP.
CPF3141 E	Member &2 not found.
CPF34C0 E	Value &1 for number of fields to return parameter not valid.
CPF3C07 E	Error occurred while retrieving information from object &1.
CPF3C1D E	Length specified in parameter &1 not valid.
CPF3C21 E	Format name &1 is not valid.
CPF3C29 E	Object name &1 is not valid.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C39 E	Value for reserved field not valid.
CPF3C3B E	Value for parameter &2 for API &1 not valid.
CPF3C82 E	Key &1 not valid for API &2.

Message ID	Error Message Text
CPF3C89 E	Key &1 specified more than once.
CPF3CE2 E	Continuation handle not valid.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF4268 E	Object &1 in &2 type *&3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9804 E	Object &2 in library &3 damaged.
CPF980B E	Object &1 in library &2 not available.
CPF9810 E	Library &1 not found.
CPF9812 E	File &1 in library &2 not found.
CPF9814 E	Device &1 not found.
CPF9820 E	Not authorized to use library &1.
CPF9822 E	Not authorized to file &1 in library &2.
CPF9825 E	Not authorized to device &1.
CPF9826 E	Cannot allocate file &2.
CPF9830 E	Cannot assign library &1
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB401 E	API &3 failed with reason code &1.
CPFB8ED E	Device description &1 not correct for operation.

Related Information

- the <**qdbst.h**> include file in library QSYSINC, for API-related structure declarations and special value declarations.
- the <**qdbstmgr.h**> include file in library QSYSINC, for the QdbstListStatistics API prototype.
- the <**qdbstls.h**> include file in library QSYSINC, for the QDBSTLS API prototype.
- "Cancel Requested Statistics Collections (QDBSTCRS, QdbstCancelRequestedStatistics) API" on page 373 (QDBSTCRS, QdbstCancelRequestedStatistics) API
- "Delete Statistics Collections (QDBSTDS, QdbstDeleteStatistics) API" on page 381 (QDBSTDS, QdbstDeleteStatistics) API
- "List Requested Statistics Collections (QDBSTLRS, QdbstListRequestedStatistics) API" on page 392 (QDBSTLRS, QdbstListRequestedStatistics) API
- "List Statistics Collection Details (QDBSTLDS, QdbstListDetailStatistics) API" on page 401 (QDBSTLDS, QdbstListDetailStatistics) API
- "Request Statistics Collections (QDBSTRS, QdbstRequestStatistics) API" on page 426 (QDBSTRS, QdbstRequestStatistics) API
- "Update Statistics Collections (QDBSTUS, QdbstUpdateStatistics) API" on page 437 (QDBSTUS, QdbstUpdateStatistics) API

API introduced: V5R2

Top | "Database and File APIs," on page 1 | APIs by category

Query SQL Database Monitor (QQQQSDBM) API

Required Parameter Group:

1	Qualified job name
Input	Char(26)
2	Number of active monitors
Output	Binary(4)
3	Size of active monitors array
Input	Binary(4)
4	Type of active monitors array
Output	Array(*) of Char(10)
5	Memory handle
Output	Char(10)
6	Error code
I/O	Char(*)

Default Public Authority: *USE

Threadsafe: Yes

The Query SQL Database Monitor (QQQSSDBM) API returns information about the activity of the SQL and the original database monitor. Associated APIs include the following:

- Clear SQL Database Monitor Statistics (QQQCSDBM)
- Dump SQL Database Monitor (QQQDSDBM)
- End SQL Database Monitor (QQQESDBM)
- Start SQL Database Monitor (QQQSSDBM)

Authorities and Locks

None

Required Parameter Group

Qualified job name
INPUT; CHAR(26)

The job for which status is being requested. The qualified job name has three parts:

Job name	CHAR(10). A specific job name, a generic name, or one of the following special values:

** or *CURRENT*
Only the job that this program is running in. The rest of the qualified job name parameter must be blank.

**ALL* All jobs. The rest of the job name parameter must be blank.

User name	CHAR(10). A specific user profile name.
Job number	CHAR(6). A specific job number.

Number of active monitors
OUTPUT; BINARY(4)

The number of active database monitors. If the number of active monitors is greater than the size of the type of active monitors array allocated by the user, the type of active monitors array is truncated to the size allocated by the user.

Size of active monitors array
INPUT; BINARY(4)

The amount of storage (number of character(10) array entries) allocated by the caller for the type of active monitors array parameter.

Type of active monitors array
OUTPUT; Array(*) of CHAR(10)

The types of database monitors that are active. The values may include:

**FILE*	The file-based database monitor (STRDBMON) is active
**SQLMEMORY*	The SQL memory-based database monitor (QQQSSDBM) is active.

Memory handle
OUTPUT; CHAR(10)

The memory handle used for the specified job if the memory-based monitor is active. Only the first 6 characters will be used for naming the memory handle.

This field is blank if the SQL memory-based database monitor is not active.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	**Error Message Text**
CPF0172 E	Parameters passed on CALL do not match those required.
CPF1321 E	job &1 user &2 job number &3 not found.
CPF3CF1 E	Error code parameter not valid.
CPF436E E	Job &1 user &2 job number &3 is not active.

API introduced: V4R3

Top | "Database and File APIs," on page 1 | APIs by category

Request Statistics Collections (QDBSTRS, QdbstRequestStatistics) API

Required Parameter Group:

1	Input data	
Input	Char(*)	
2	Length of input data	
Input	Binary(4)	
3	Format of input data	
Input	Char(8)	
4	Feedback area	
Output	Char(*)	
5	Length of feedback area	
Input	Binary(4)	
6	Feedback keys	
Input	Array(*) of Binary(4)	
7	Number of feedback keys	
Input	Binary(4)	
8	Error code	
I/O	Char(*)	

Service Program Name: QDBSTMGR

Default Public Authority: *USE

Threadsafe: Yes

The Request Statistics Collections (QDBSTRS, QdbstRequestStatistics) API allows the user to request one or more statistics collections for a given set of columns of a database file member to be created.

The created statistics collections are stored as part of the database file member.

Options are provided to control whether the statistics are generated in the background or to be processed immediately, and whether the status of the resulting statistics collections is to be maintained automatically by the system or manually by the user.

Section overview

- “Related Information” on page 433

Authorities and Locks

ASP Device Authority
: *EXECUTE

File Authority
: *OBJALTER, *OBJOPR

File Library Authority
: *EXECUTE

File Lock
: *SHRRD

Translation Table Authority
: *USE

Translation Table Library Authority
: *EXECUTE

Translation Table Lock
: *SHRRD

Required Parameter Group

Input data
INPUT; CHAR(*)

The buffer containing the input parameters according to the *Format of input data* parameter. The buffer content has to start at a four-byte boundary.

Length of input data
INPUT; BINARY(4)

The length of the input data buffer provided.

Format of input data
INPUT; CHAR(8)

The format of the input data. Possible values are:

STIR0100	Basic request statistics collections input parameters.

Refer to “STIR0100 Input Format” on page 428 for more information.

Feedback area
OUTPUT; CHAR(*)

The buffer to receive feedback data. See “Feedback Area Format” on page 429 for more information. The buffer content has to start at a four-byte boundary.

Length of feedback area
INPUT; BINARY(4)

The length of the feedback area buffer provided. The required minimum length is 16, to fit the feedback area header (see “Feedback Area Format” on page 429).

Feedback keys
INPUT; ARRAY(*) OF BINARY(4)

The list of fields to return in the feedback area. For a list of valid keys see “Valid Keys - Feedback” on page 429.

Number of feedback keys
INPUT; BINARY(4)

The number of fields to return in the feedback area. If zero is specified, all other feedback area parameters (*Feedback area*, *Length of feedback area*, and *Feedback keys*) are ignored.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

STIR0100 Input Format

The basic request statistics collections input parameters. See "Field Descriptions" on page 429 for details of the fields listed.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	ASP device name
10	A	CHAR(10)	File name
20	14	CHAR(10)	File library name
30	1E	CHAR(10)	File member name
40	28	CHAR(12)	Collection mode
52	34	BINARY(4)	Offset to statistics collections
56	38	BINARY(4)	Number of statistics collections
60	3C	CHAR(*)	Reserved

The fields below follow the fields above and repeat, in the order listed, for each statistics collection, where the first statistics collection starts at the given *Offset to statistics collections*.

Offset			
Dec	**Hex**	**Type**	**Field**
+0	+0	BINARY(4)	Length of statistics collection
+4	+4	BINARY(4)	Length of statistics collection name
+8	+8	CHAR(128)	Statistics collection name
+136	+88	CHAR(10)	Aging mode
+146	+92	CHAR(2)	Reserved
+148	+94	BINARY(4)	Displacement to columns
+152	+98	BINARY(4)	Number of columns
+156	+9C	CHAR(*)	Reserved

The fields below follow for each statistics collection definition header structure as described above and repeat, in the order listed, for each column in the current statistics collection, where the data for the first column starts at the given offset *Offset to columns*.

Offset			
Dec	**Hex**	**Type**	**Field**
+0	+0	BINARY(4)	Length of column definition
+4	+4	CHAR(10)	Column name

Offset			
Dec	Hex	Type	Field
+14	+E	CHAR(10)	Translation table name
+24	+18	CHAR(10)	Translation table library name
+34	+22	CHAR(2)	Reserved
+36	+24		

Valid Keys - Feedback

Use the following keys to specify the fields to be returned in the feedback area. Each key can only be specified once. See "Field Descriptions" for details of the fields listed.

Key	Type	Description
1	CHAR(10)	ASP device name used
3	CHAR(10)	File library name used
4	CHAR(10)	File member name used
8	BINARY(4)	Elapsed time
6	CHAR(16)	Internal request ID
43	BINARY(4)	Total number of statistics collections for internal request ID
46	Array of CHAR(*)	Statistics collection names used
7	Array of CHAR(*)	Internal statistics IDs created

Feedback Area Format

The fields returned in the feedback area will be returned in the order requested. See "Field Descriptions" for details of the fields listed.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	BINARY(4)	Number of key fields returned
12	C	BINARY(4)	Number of key fields available
These fields repeat, in the order listed, for each key selected.		BINARY(4)	Length of field information returned
		BINARY(4)	Key identifier
		BINARY(4)	Length of data
		CHAR(*)	Data
		CHAR(*)	Reserved (padding to the next 4 bytes boundary)

Field Descriptions

Aging mode. Whether the system is allowed to age or remove the resulting collected statistics collection. The possible values are:

**SYS*	Refresh or removal of the resulting statistics collections will be performed automatically by the system.

**USER*	Refresh or removal will only occur when a user requests it.

ASP device name. The name of one auxiliary storage pool (ASP) device in the ASP group in which the library and file are located. The ASP device must have a status of 'Available'. The documented authority is required for the given ASP and the primary of the corresponding ASP group. The name can be a specific ASP device name (for an ASP with a number greater than 32), or one of the following special values:

***	Locate the library and file in the name space for the current thread.
**SYSBAS*	Locate the library and file in the system ASP (ASP number 1) and all basic ASPs (ASP numbers 2 through 32).

ASP device name used. The actual auxiliary storage pool device name used, after possible resolution of special values.

Collection mode. Where the processing for the statistics collection will be performed, or if merely an estimate is requested. The possible values are:

**IMMEDIATE*	Execute the request immediately. The statistics collection will run in the user's process. Control will not return to the API invoker until the collection is complete.
**BACKGROUND*	The statistics collection will be scheduled for execution in system job QDBFSTCCOL. Control will return to the API invoker immediately. **Note:** If the current setting of the system value QDBFSTCCOL does not allow user requested background collections, then the request will be queued until the system value is changed to a level allowing the execution of the request.
**ESTIMATE*	An estimate is returned immediately for the time, that would be required to run the statistics collection. No statistics collection will actually be created.

Column name. The name of a single column within a single statistics collection definition.

Data. The data returned for the key identifier.

Displacement to columns. Displacement to the start of the list of column definitions for the current statistics collection definition.

Elapsed time. When the collection mode specified is **IMMEDIATE*, the value represents the number of seconds actually spent processing the requested statistics collection.

For any other collection mode, this value represents the estimated time in seconds, that the statistics collection should take.

File library name. Where the file for which statistics collections are being requested is located. You can use these special values for the library name, if the *ASP Device Name* is *:

**CURLIB*	The job's current library or QGPL if the current library is not set.
**LIBL*	The library list.
**USRLIBL*	Libraries listed in the user portion of the library list.

File library name used. The actual file library name used, after possible resolution of special values.

File member name. The name of the file member to be used for the statistics collections request.

This value can be a specific file member name or one of the following special values:

**FIRST*	The first member (in the order created) in the specified file.
**LAST*	The last member (in the order created) in the specified file.

File member name used. The actual file member name used, after possible resolution of special values.

File name. The name of the file for which statistics collections are being requested. The file has to be an existing local, single format, physical file.

Internal request ID. For a *Collection mode* of **BACKGROUND* only, this field is an unique ID for the complete list of statistics collections requested here. The request ID stays valid until the request is completed and the ID can be used on the "Cancel Requested Statistics Collections (QDBSTCRS, QdbstCancelRequestedStatistics) API" on page 373 (QDBSTCRS, QdbstCancelRequestedStatistics) API.

Note: The ID is stored in binary, non printable form in the character array.

Internal statistics IDs created. For a *Collection mode* of **IMMEDIATE* only, this will return an array of the internal statistics ID created for each of the requested and successfully created statistics collections. This statistics ID together with the qualified member name can serve as a unique identifier for the created statistics collection on the input to the "Delete Statistics Collections (QDBSTDS, QdbstDeleteStatistics) API" on page 381, "Update Statistics Collections (QDBSTUS, QdbstUpdateStatistics) API" on page 437, and "List Statistics Collection Details (QDBSTLDS, QdbstListDetailStatistics) API" on page 401 APIs. Each ID is returned as a single key value. The single key values will be returned in sequence and in the order the statistics collections were requested in the input format. The array dimension can be determined either by the input format field *Number of statistics collections* or by requesting this number again as *Total number of statistics collections for internal request ID* in the feedback.

Note: The internal statistics ID for a statistics collection is also returned on the "List Statistics Collections (QDBSTLS, QdbstListStatistics) API" on page 412 API.

Note: The ID is stored in binary, non printable form in the character array.

Key identifier. The field returned. For a list of valid keys see "Valid Keys - Feedback" on page 429.

Length of column definition. Length of this column definition.

Length of data. The length of the data returned for the field.

Length of field information returned. Total number of bytes returned for this field in the feedback area.

Length of statistics collection. Length of this statistics collection definition, which can be used to get to the next definition.

Note: The length includes all the column definitions for this statistics collections.

Length of statistics collection name. Actual length of the statistics collection name, up to the maximum length of 128 characters.

Number of bytes available. Number of bytes that could be returned in the feedback area.

Number of bytes returned. Number of bytes returned in the feedback area.

Number of columns. Number of columns in a single statistics collection definition.

Note: This number must be 1.

Number of key fields available. Number of fields that could be returned in the feedback area.

Number of key fields returned. Number of fields returned in the feedback area.

Number of statistics collections. Number of statistics collection definitions for this request.

Offset to statistics collections. Offset to the start of the list of statistics collection definitions for this request.

Reserved. Reserved for future use. If this field is input, the field must be set to hexadecimal zeros.

Reserved (in feedback area format). Structure padding to guarantee alignment to the next four bytes boundary.

Statistics collection names used. An array of the statistics collection names used. Each name is returned as a single key value and is either the name specified on input, or the system generated name, if the special value **GEN* was used on input and if the *Collection mode* specified is **IMMEDIATE*. The single key values will be returned in sequence and in the order the statistics collections were requested in the input format. The array dimension can be determined either by the input format field *Number of statistics collections* or by requesting this number again as *Total number of statistics collections for internal request ID* in the feedback.

Statistics collection name. A name unique amongst all statistics collections for the file member. The following special value can be used:

**GEN*	The system will generate a unique name for the statistics collection.

Note: The name is given in varying length form, where the actual length is passed in the *Length of statistics collection name* field, to indicate how many of the 128 characters are actually part of the name text.

Total number of statistics collections for internal request ID. Number of statistics collection definitions for this request. Gives the array dimension of *Statistics collection names used* and *Internal statistics IDs created* and is a copy of *Number of statistics collections* in the input format.

Translation table name. This field is relevant just for character columns and must be all blanks otherwise. For character columns, this is the name of a translation table to be applied to the data in this column. The name must be for an existing translation table, or all blanks, if no translation table is to be applied.

Translation table library name. Where the translation table is located. The name must be for an existing library or all blanks, if no translation table is to be applied.

You can use these special values for the library name:

**CURLIB*	The job's current library or QGPL if the current library is not set.
**LIBL*	The library list.
**USRLIBL*	Libraries listed in the user portion of the library list.

Error Messages

Message ID	**Error Message Text**
CPF0623 E	Field &1 not found in record format &2.
CPF1866 E	Value &1 for number of fields to return not valid.

Message ID	Error Message Text
CPF2105 E	Object &1 in &2 type *&3 not found.
CPF2113 E	Cannot allocate library &1.
CPF2173 E	Value for ASPDEV not valid with special value for library.
CPF218C E	&1 not a primary or secondary ASP.
CPF3141 E	Member &2 not found.
CPF34C0 E	Value &1 for number of fields to return parameter not valid.
CPF3C07 E	Error occurred while retrieving information from object &1.
CPF3C1D E	Length specified in parameter &1 not valid.
CPF3C21 E	Format name &1 is not valid.
CPF3C29 E	Object name &1 is not valid.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C39 E	Value for reserved field not valid.
CPF3C3B E	Value for parameter &2 for API &1 not valid.
CPF3C82 E	Key &1 not valid for API &2.
CPF3C89 E	Key &1 specified more than once.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF4268 E	Object &1 in &2 type *&3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9804 E	Object &2 in library &3 damaged.
CPF980B E	Object &1 in library &2 not available.
CPF9810 E	Library &1 not found.
CPF9812 E	File &1 in library &2 not found.
CPF9814 E	Device &1 not found.
CPF9820 E	Not authorized to use library &1.
CPF9822 E	Not authorized to file &1 in library &2.
CPF9825 E	Not authorized to device &1.
CPF9826 E	Cannot allocate file &2.
CPF9830 E	Cannot assign library &1
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB401 E	API &3 failed with reason code &1.
CPFB8ED E	Device description &1 not correct for operation.

Related Information

- the <**qdbst.h**> include file in library QSYSINC, for API-related structure declarations and special value declarations.
- the <**qdbstmgr.h**> include file in library QSYSINC, for the QdbstRequestStatistics API prototype.
- the <**qdbstrs.h**> include file in library QSYSINC, for the QDBSTRS API prototype.
- the system value QDBFSTCCOL.
- “Cancel Requested Statistics Collections (QDBSTCRS, QdbstCancelRequestedStatistics) API” on page 373 (QDBSTCRS, QdbstCancelRequestedStatistics) API
- “Delete Statistics Collections (QDBSTDS, QdbstDeleteStatistics) API” on page 381 (QDBSTDS, QdbstDeleteStatistics) API
- “List Requested Statistics Collections (QDBSTLRS, QdbstListRequestedStatistics) API” on page 392 (QDBSTLRS, QdbstListRequestedStatistics) API
- “List Statistics Collection Details (QDBSTLDS, QdbstListDetailStatistics) API” on page 401 (QDBSTLDS, QdbstListDetailStatistics) API
- “List Statistics Collections (QDBSTLS, QdbstListStatistics) API” on page 412 (QDBSTLS, QdbstListStatistics) API

- "Update Statistics Collections (QDBSTUS, QdbstUpdateStatistics) API" on page 437 (QDBSTUS, QdbstUpdateStatistics) API

API introduced: V5R2

Top | "Database and File APIs," on page 1 | APIs by category

Start SQL Database Monitor (QQQSSDBM) API

Required Parameter Group:

1	Qualified job name
Input	Char(26)
2	Memory handle
Input	Char(10)
3	Storage size
Input	Binary(4)
4	Free storage method
Input	Char(10)
5	Number of types to monitor
Input	Binary(4)
6	Subtypes to monitor
Input	Array(*) of Char(10)
7	Error code
I/O	Char(*)

Default Public Authority: *USE

Threadsafe: Conditional; see "Usage Notes" on page 436.

The Start SQL Database Monitor (QQQSSDBM) API starts the memory-based SQL database monitor. Associated APIs include the following:

- Clear SQL Database Monitor Statistics (QQQCSDBM)
- Dump SQL Database Monitor (QQQDSDBM)
- End SQL Database Monitor (QQQESDBM)
- Query SQL Database Monitor (QQQQSDBM)

Authorities and Locks

Current User Profile
*JOBCTL

Required Parameter Group

Qualified job name
INPUT; CHAR(26)

The job to be monitored. The qualified job name has three parts:

Job name	CHAR(10). A specific job name, a generic name, or one of the following special values:
	** or *CURRENT* Only the job that this program is running in. The rest of the qualified job name parameter must be blank.
	**ALL* All jobs. The rest of the job name parameter must be blank.
User name	CHAR(10). A specific user profile name.
Job number	CHAR(6). A specific job number.

Memory handle
INPUT; CHAR(10)

The handle used for consolidating data. This parameter is only valid when the qualified job name parameter is not *ALL (that is, you are starting the monitor on a specific job). If multiple jobs are monitoring with the same memory handle, their database activity will be consolidated together.

If *JOB is specified, the job's database activity will be monitored in its own memory area (the activity will not be consolidated with any other job's database activity, unless the other job explicitly specifies this job's job number as the memory handle.). For example, assume QQQSSDBM is issued by job 111111 with a memory handle of *JOB. This implies a memory handle of 111111 is used. If job 999999 issues QQQSSDBM and names a memory handle of 111111, then both jobs 111111 and 999999 will use memory area 111111. Consequently, the database monitor data for both jobs will be summarized within this memory area.

The possible values are:

User defined	Up to 6-character value to name a memory area that will contain consolidated data. Only the first 6 characters will be used for a named memory.
**JOB*	Use the memory area associated with the job, and do not consolidate data with any other job.

Storage size
INPUT; BINARY(4)

The maximum amount of storage to use for in-memory data (specified in megabytes). A value of -1 implies no maximum.

Free storage method
INPUT; CHAR(10)

When maximum storage is reached in the storage size parameter, the method used to free storage. The possible value is:

**LRU*	Free the statement least recently used

Number of types to monitor
INPUT; BINARY(4)

The number of types passed in the subtypes to monitor array.

Subtypes to monitor
INPUT; Array(*) of CHAR(10)

The list of all subtypes that should be monitored. The possible values are:

KEYT_3000	Summary: Arrival sequence (file QAQQ3000)
KEYT_3001	Summary: Index used (file QAQQ3001)
KEYT_3002	Summary: Index created (file QAQQ3002)

KEYT_3003	Summary: Sort (file QAQQ3003)
KEYT_3004	Summary: Temporary file (file QAQQ3004)
KEYT_3007	Summary: Optimizer time-out or all access paths considered (file QAQQ3007)
KEYT_3008	Summary: Subselect processing (file QAQQ3008)
KEYT_3010	Summary: Host variable values (file QAQQ3010)
KEYT_TEXT	SQL statement text (file QAQQTEXT)
KEYT_QRYI	Summary: General SQL information including statement count, maximum runtime, time last used, and so forth. This subtype is always monitored because it is required for monitoring all other subtypes (file QAQQQRYI). It should still be specified, and is required if it is the only subtype to be monitored.
**EDSQL*	Monitor all subtypes. If this option is specified, the number of types to monitor should be set to 1, and no other subtypes should be requested.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Usage Notes

This function is threadsafe but not thread enabled. Database monitor data is collected in the threaded process but summarized at the job level.

Error Messages

Message ID	**Error Message Text**
CPD0172 D	Parameters passed on CALL do not match those required.
CPF1321 E	Job &1 user &2 job number &3 not found.
CPF222E E	&1 special authority is required.
CPF3CF1 E	Error code parameter not valid.
CPF436C E	Job specified is already being monitored.
CPF436E E	Job &1 user &2 job number &3 is not active.

API introduced: V4R3

Top | “Database and File APIs,” on page 1 | APIs by category

Update Statistics Collections (QDBSTUS, QdbstUpdateStatistics) API

Required Parameter Group:

1	Input data	
Input	Char(*)	
2	Length of input data	
Input	Binary(4)	
3	Format of input data	
Input	Char(8)	
4	Feedback area	
Output	Char(*)	
5	Length of feedback area	
Input	Binary(4)	
6	Feedback keys	
Input	Array(*) of Binary(4)	
7	Number of feedback keys	
Input	Binary(4)	
8	Error code	
I/O	Char(*)	

Service Program Name: QDBSTMGR

Default Public Authority: *USE

Threadsafe: Yes

The Update Statistics Collection (QDBSTUS, QdbstUpdateStatistics) API allows the user to update the attributes and to refresh the data of an existing single statistics collection.

In addition, the QdbstUpdateStatistics API allows to block all future system initiated statistics collection requests for a specific database file member.

Section overview

Authorities and Locks

ASP Device Authority
*EXECUTE

File Authority
*OBJALTER, *OBJOPR

File Library Authority
*EXECUTE

File Lock
*SHRRD

Required Parameter Group

Input data
INPUT; CHAR(*)

The buffer containing the input parameters according to the format of input data parameter. The buffer content has to start at a four-byte boundary.

Length of input data
INPUT; BINARY(4)

The length of the input data buffer provided.

Format of input data
INPUT; CHAR(8)

The format of the input data. Possible values are:

STIU0100	Update statistics collection via unique statistics ID and keyed input.

Refer to "STIU0100 Input Format" on page 439 for more information.

Feedback area
OUTPUT; CHAR(*)

The buffer to receive feedback data. See "Feedback Area Format" on page 440 for more information. The buffer content has to start at a four-byte boundary.

Length of feedback area
INPUT; BINARY(4)

The length of the feedback area buffer provided. The required minimum length is 16, to fit the feedback area header (see "Feedback Area Format" on page 440).

Feedback keys
INPUT; ARRAY(*) OF BINARY(4)

The list of fields to return in the feedback area. For a list of valid keys see "Valid Keys - Feedback" on page 439.

Number of feedback keys
INPUT; BINARY(4)

The number of fields to return in the feedback area. If zero is specified, all other feedback area parameters (*Feedback area*, *Length of feedback area*, and *Feedback keys*) are ignored.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

STIU0100 Input Format

Update statistics collection input parameters. See "Field Descriptions" on page 440 for details of the fields listed.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	ASP device name
10	A	CHAR(10)	File name
20	14	CHAR(10)	File library name
30	1E	CHAR(10)	File member name
40	28	CHAR(16)	Internal statistics ID
56	38	BINARY(4)	Offset to fields to update
60	3C	BINARY(4)	Number of fields to update
64	40	CHAR(*)	Reserved
These fields repeat, in the order listed, for each field to be updated, started at the given offset.		BINARY(4)	Length of field information
		BINARY(4)	Key identifier
		BINARY(4)	Length of data
		CHAR(*)	Data
		CHAR(*)	Reserved (padding to the next 4-byte boundary)
		CHAR(*)	Reserved

Valid Keys - Update

Use the following keys to specify the fields to be updated when using the "STIU0100 Input Format." Each key can only be specified once. See "Field Descriptions" on page 440 for details of the fields listed.

Key	Type	Description
45	CHAR(12)	Statistics data (key value is the collection mode)
18	CHAR(10)	Aging mode
46	CHAR(*)	Statistics collection name
47	CHAR(1)	Block system statistics collections option

Valid Keys - Feedback

Use the following keys to specify the fields to be returned in the feedback area. Each key can only be specified once. See "Field Descriptions" on page 440 for details of the fields listed.

Key	Type	Description
1	CHAR(10)	ASP device name used
3	CHAR(10)	File library name used
4	CHAR(10)	File member name used
6	CHAR(16)	Internal request ID
18	CHAR(10)	Previous aging mode
46	CHAR(*)	Previous statistics collection name
47	CHAR(1)	Previous block system statistics collections option

Feedback Area Format

The fields returned in the feedback area are returned in the order requested. See "Field Descriptions" for details of the fields listed.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	BINARY(4)	Number of key fields returned
12	C	BINARY(4)	Number of key fields available
These fields repeat, in the order listed, for each key selected.		BINARY(4)	Length of field information returned
		BINARY(4)	Key identifier
		BINARY(4)	Length of data
		CHAR(*)	Data
		CHAR(*)	Reserved (padding to the next 4 bytes boundary)

Field Descriptions

Aging mode. Whether the system is allowed to age or remove the statistics collection. The possible values to change to are:

**SYS*	Refresh or removal of the statistics collections will be performed automatically by the system.
**USER*	Refresh or removal will only occur when a user requests it.

ASP device name. The name of one auxiliary storage pool (ASP) device in the ASP group in which the library and file are located. The ASP device must have a status of 'Available'. The documented authority is required for the given ASP and the primary of the corresponding ASP group. The name can be a specific ASP device name (for an ASP with a number greater than 32), or one of the following special values:

*	Locate the library and file in the name space for the current thread.
**SYSBAS*	Locate the library and file in the system ASP (ASP number 1) and all basic ASPs (ASP numbers 2 through 32).

ASP device name used. The actual auxiliary storage pool device name used, after possible resolution of special values.

Block system statistics collections option. Whether future system initiated (automatic) statistics collection create or update requests will be allowed for this database file member. The possible values are:

'0'	Do not block system initiated statistics collection requests. **Note:** This is the system default.
'1'	Block system initiated statistics collection requests.

Note: The internal statistics ID is ignored for this option, which operates at file member level, but the ID has to be a valid statistics ID, if any other update option besides the block option is specified.

Note: Currently active system requests will not be affected by changing this option. See “Cancel Requested Statistics Collections (QDBSTCRS, QdbstCancelRequestedStatistics) API” on page 373.

Data (in feedback area format). The data returned for the key identifier.

Data (in STIU0100 input format). The data with which the field is to be updated.

File library name. Where the file for which statistics collection attributes are to be updated is located. You can use these special values for the library name, if the *ASP Device Name* is *:

**CURLIB*	The job’s current library or QGPL if the current library is not set.
**LIBL*	The library list.
**USRLIBL*	Libraries listed in the user portion of the library list.

File library name used. The actual file library name used, after possible resolution of special values.

File member name. The name of the file member to be used for the statistics collection update request.

This value can be a specific file member name or one of the following special values:

**FIRST*	The first member (in the order created) in the specified file.
**LAST*	The last member (in the order created) in the specified file.

File member name used. The actual file member name used, after possible resolution of special values.

File name. The name of the file for which statistics collection attributes are to be updated. The file has to be an existing local, single format, physical file.

Internal request ID. If the update key *Statistics Data* is specified and its value is **BACKGROUND*, this field is an unique ID for the statistics data update requested here. The request ID stays valid until the statistics data update is completed and the ID can be used on the “Cancel Requested Statistics Collections (QDBSTCRS, QdbstCancelRequestedStatistics) API” on page 373 (QDBSTCRS, QdbstCancelRequestedStatistics) API.

Note: The ID is stored in binary, non printable form in the character array.

Internal statistics ID. Together with the qualified file name and member name, this represents a unique ID for the statistics collection to be updated. See “List Statistics Collections (QDBSTLS, QdbstListStatistics) API” on page 412 and Request statistics collections (OPM, QDBSTRS; ILE QdbstRequestStatistics) API.

Note: The ID is stored in binary, non printable form in the character array.

Key identifier (in the STIU0100 input format). The field to be updated. For a list of valid keys see “Valid Keys - Update” on page 439.

Key identifier (in the feedback area format). The field returned. For a list of valid keys see “Valid Keys - Feedback” on page 439.

Keys of fields to update. The list of fields to update per list entry. For a list of valid keys see “Valid Keys - Update” on page 439.

Length of data (in feedback area format). The length of the data returned for the field.

Length of data (in STIU0100 input format). The length of the data the field is to be updated with.

Length of field information. Total number of bytes being passed for the field to be updated.

Length of field information returned. Total number of bytes returned for this field in the feedback area.

Number of bytes available. Number of bytes that could be returned in the feedback area.

Number of bytes returned. Number of bytes returned in the feedback area.

Number of key fields available. Number of fields that could be returned in the feedback area.

Number of key fields returned. Number of fields returned in the feedback area.

Number of fields to update. The number of fields to update.

Offset to fields to update. Offset to the start of the array of fields to update.

Previous aging mode. The aging mode in effect before the update.

Note: If the aging mode was not requested to be updated, the aging mode returned will be blank.

Previous block system statistics collections option. The block option in effect before the update.

Note: If the block option was not requested to be updated, the block option returned will be blank.

Previous statistics collection name. The name in effect before the update.

Note: If the statistics collection name was not requested to be updated, the statistics collection name returned will have a length of 0.

Reserved. Reserved for future use. If this field is input, the field must be set to hexadecimal zeros.

Reserved (in feedback area format). Structure padding to guarantee alignment to the next four bytes boundary.

Reserved (in STIU0100 input format). Structure padding to guarantee alignment to the next four bytes boundary.

Statistics collection name. A name unique amongst all statistics collections for the file member.

Statistics data. The statistics data is to be refreshed. The key value is the collection mode (see also the "Request Statistics Collections (QDBSTRS, QdbstRequestStatistics) API" on page 426), under which the refresh should take place. The possible values are:

**IMMEDIATE*	Execute the refresh immediately. The statistics collection will run in the user's process. Control will not return to the API invoker until the refresh is complete.
**BACKGROUND*	The refresh will be scheduled for execution in system job QDBFSTCCOL. Control will return to the API invoker immediately. **Note:** If the current setting of the system value QDBFSTCCOL does not allow user requested background collections, then the refresh request will be queued until the system value is changed to a level allowing the execution of the refresh.

Error Messages

Message ID	Error Message Text
CPF0623 E	Field &1 not found in record format &2.
CPF1866 E	Value &1 for number of fields to return not valid.
CPF2105 E	Object &1 in &2 type *&3 not found.
CPF2113 E	Cannot allocate library &1.
CPF2173 E	Value for ASPDEV not valid with special value for library.
CPF218C E	&1 not a primary or secondary ASP.
CPF3141 E	Member &2 not found.
CPF34C0 E	Value &1 for number of fields to return parameter not valid.
CPF3C07 E	Error occurred while retrieving information from object &1.
CPF3C1D E	Length specified in parameter &1 not valid.
CPF3C21 E	Format name &1 is not valid.
CPF3C29 E	Object name &1 is not valid.
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3C39 E	Value for reserved field not valid.
CPF3C3B E	Value for parameter &2 for API &1 not valid.
CPF3C82 E	Key &1 not valid for API &2.
CPF3C89 E	Key &1 specified more than once.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF4268 E	Object &1 in &2 type *&3 not found.
CPF9802 E	Not authorized to object &2 in &3.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9804 E	Object &2 in library &3 damaged.
CPF980B E	Object &1 in library &2 not available.
CPF9810 E	Library &1 not found.
CPF9812 E	File &1 in library &2 not found.
CPF9814 E	Device &1 not found.
CPF9820 E	Not authorized to use library &1.
CPF9822 E	Not authorized to file &1 in library &2.
CPF9825 E	Not authorized to device &1.
CPF9826 E	Cannot allocate file &2.
CPF9830 E	Cannot assign library &1
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPFB401 E	API &3 failed with reason code &1.
CPFB8ED E	Device description &1 not correct for operation.

Related Information

- the <**qdbst.h**> include file in library QSYSINC, for API-related structure declarations and special value declarations.
- the <**qdbstmgr.h**> include file in library QSYSINC, for the QdbstUpdateStatistics API prototype.
- the <**qdbstus.h**> include file in library QSYSINC, for the QDBSTUS API prototype.
- the system value QDBFSTCCOL.
- "Cancel Requested Statistics Collections (QDBSTCRS, QdbstCancelRequestedStatistics) API" on page 373 (QDBSTCRS, QdbstCancelRequestedStatistics) API
- "Delete Statistics Collections (QDBSTDS, QdbstDeleteStatistics) API" on page 381 (QDBSTDS, QdbstDeleteStatistics) API
- "List Requested Statistics Collections (QDBSTLRS, QdbstListRequestedStatistics) API" on page 392 (QDBSTLRS, QdbstListRequestedStatistics) API

- “List Statistics Collection Details (QDBSTLDS, QdbstListDetailStatistics) API” on page 401 (QDBSTLDS, QdbstListDetailStatistics) API
- “List Statistics Collections (QDBSTLS, QdbstListStatistics) API” on page 412 (QDBSTLS, QdbstListStatistics) API
- “Request Statistics Collections (QDBSTRS, QdbstRequestStatistics) API” on page 426 (QDBSTRS, QdbstRequestStatistics) API

API introduced: V5R2

Top | “Database and File APIs,” on page 1 | APIs by category

Visual Explain (QQQVEXPL) API

Required Parameter Group:

1	Pointer to qualified query name
Input	CHAR(*)
2	Pointer to qualified database monitor table
Input	CHAR(*)
3	Pointer to the set of records returned
I/O	PTR(SPP)
4	Pointer to the return code structure
I/O	CHAR(*)

Threadsafe: Conditional; see “Context Types” on page 452.

The Visual Explain (QQQVEXPL) API is used to create a query graph that graphically displays the execution of an SQL statement. You can use this tool to see information about both static and dynamic SQL statements. QQQVEXPL supports the following types of SQL statements:
- Select
- Insert
- Update
- Delete

You can use this tool to better understand where the highest costs of your queries are taking place. You can improve query performance by:
- Rewriting your SQL statement.
- Changing query attributes and environment settings.
- Creating any recommended indexes.

You also can use the QQQVEXPL API to:
- View the statistics that were used at the time of optimization.
- Determine whether an index was used to access a table. If an index was not used, Visual Explain can help you determine which columns might benefit from being indexed.
- View the effects of performing various tuning techniques by comparing the before and after versions of the query graph.
- Obtain information about each operation in the query graph, including the total estimated cost and number of rows retrieved.

Input to the Visual Explain (QQQVEXPL) API is two structures. One contains the information the Visual Explain consolidator needs to uniquely identify which query within the database monitor table is to be explained. The other contains the name of the database monitor table. The database monitor table is a table that contains the records resulting from an execution of the STRDBMON command. Output from the Visual Explain (QQQVEXPL) API is a pointer to a stream of data located in user domain storage. This data contains the information necessary to create a pictorial view of how the specified query was implemented. It is up to the user to clean up the user domain storage. Also, output is a structure that contains an error return code, the number of entries in the output data, and the entry number of the Final Select ICON. To create the picture, the user starts with the entry of the Final Select ICON and works back to the beginning ICONs.

The format for the output records (or array entries) can be found in "Output Format" on page 448. Each record has a unique I CON number associated with it. The unique ICON number associates the records to a particular ICON. That is, all records with the same unique ICON number are associated with one specific ICON. For example, if the Final Select ICON has a unique ICON number of 12, then all records with a unique ICON number of 12 contain information about the Final Select ICON. The record immediately following the Final Select record is the record that tells the user how many ICONs (called child ICONs) are branched off the Final Select ICON. This record will have a record type of 11, which means it contains the number of child ICONs. The unique ICON number will match the unique ICON number of the Final Select ICON. Therefore, we know this record is telling us how many child ICONs there are for the Final Select ICON. The next records will contain the ICON number of the child ICONs. There will be one record for each child ICON and they will have a record type of 12 (unique ICON number of the child ICON). The user can find the record that corresponds to the child ICON by searching for the record that has a record type of 10 (new ICON) and a unique ICON number that matches the ICON number of the child ICON. Once the record of the child ICON is found, the process starts over again. All the records associated with that ICON (that is, that have the same unique ICON number) are read and processed. Any child ICONs are put on a stack or queue to be processed next. To see the list of possible record types, see "Record Types" on page 453.

The heart of the picture that is generated is the ICONs. In general, each ICON represents an operation performed during the execution of the query. It is up to the user to create and design the ICONs to be used. The connection between the output data and the user's ICONs is the label of the ICON that is returned within the new ICON record (record type of 10). The user is expected to match the non-translated label that is returned to the label that corresponds to the specific ICON. The non-translated ICON label is returned in the character output field. The translated ICON label is returned in the column heading field. For a list of ICON labels, see "ICON Labels" on page 450. For a detailed description of the operation represented by each ICON, see Database Performance and Query Optimization in the iSeries Information Center.

Authorities and Locks

Library Authority
*EXECUTE

Table Authority
*OBJOPR, *READ

Required Parameter Group

Pointer to the qualified query
INPUT; CHAR(*)

A pointer to a variable length structure that is used to determine the query to be explained. The structure contains two variables:

Type	**Description**
BINARY(2)	Length of the structure that contains the qualified query.

Type	Description
CHAR(*)	Structure used to determine the specific query to be explained. This structure contains seven variables. Generally, these variables are set to the same value as the corrsponding variables in the QQJFLD field within the QQQ1000 record of the query to be explained. One way to find the appropriate QQQ1000 record within the database monitor table is to view the SQL statement text (field QQ1000) and compare it to the SQL statement text of the query you wish to have explained.

System name
CHAR(8). A specific iSeries name. It is set to the same value as the QQSYS field or the first 8 bytes of field QQJFLD.

Job name
CHAR(10). A specific job name. It is set to the same value as the QQJOB field or bytes 9 to 18 of field QQJFLD.

Job user CHAR(10). A specific user profile name. It is set to the same value as the QQUSER field or bytes 19 to 28 of field QQJFLD.

Job number
CHAR(6). A specific job number. It is set to the same value as the QQJNUM field or bytes 29 to 34 of field QQJFLD.

Unique query count
BINARY(4). A unique query number. It is set to the same value as found in bytes 35 to 38 of field QQJFLD (that is, Select hex(substr(QQJFLD,35,4)) From Montable).

Statement number
BINARY(4). A specific statement number. It is set to the same value as found in bytes 39 to 42 of field QQJFLD (that is, Select hex(substr(QQJFLD,39,4)) From Montable).

Query Definition Template (QDT) number
BINARY(4). A specific QDT number. It is set to the same value as found in bytes 43 to 46 of field QQJFLD (that is, Select hex(substr(QQJFLD,43,4)) From Montable).

Pointer to qualified monitor table
INPUT; CHAR(*)

A pointer to a CHAR(72) structure containing the name of the database monitor table and other optional variables. The structure contains nine variables:

Type	Variable	Description
CHAR(10)	Monitor table name	Name of the database monitor table that contains the query to be explained.
CHAR(10)	Monitor library name	Library of the database monitor table.
CHAR(3)	Date format	A specific date format or blanks. I f blank, the date format of the current job will be extracted and used. Possible date formats are: • USA • ISO • EUR • JIS • MDY • DMY • YMD • JUL

Type	Variable	Description
CHAR(1)	Date separator	A specific date separator or J. If J, the date separator of the current job will be extracted and used. Possible date separators are: • "/" • "-" • "." • "," • " "
CHAR(3)	Time format	A specific time format. It must be one of the following values: • USA • ISO • EUR • JIS • HMS
CHAR(1)	Time separator	A specific time separator or J. If J, the time separator of the current job will be extracted and used. Possible time separators are: • ":" • "." • "," • " "
CHAR(1)	Decimal point	A specific decimal point, J, or blank. If J or blank, the decimal point of the current job will be extracted and used. Possible decimal points are: • "." • ","
CHAR(3)	Language ID	A specific language ID, J, or blanks. If J or blanks, the language ID of the current job will be extracted and used. Currently, the language ID is not used and it is recommended this value be set to blanks.
CHAR(40)	Reserved	Open for future expansion. These should be set to hexadecimal zeros.

Pointer to output data
I/O; PTR(SPP)

A pointer to data that can be viewed as a set of records or multiple entries within an array. This data is used to determine the pictorial representation of the query. The user can retrieve the data in any manner. One suggested method is to view the returned data as a set of records and use the SET RESULTS SETS command within an SQL procedure to retrieve the output data. To see the format of the output data, see "Output Format" on page 448. Once finished, it is up to the user to deallocate or destroy the space containing the output data.

Pointer to output return code
I/O; CHAR(*)

Pointer to a CHAR(32) structure that contains the following output information:

Type	Variable	Description
BINARY(4)	Error code	Error code returned from the Visual Explain consolidator. See "Error Codes" on page 457 for a list of possible return codes.
BINARY(4)	Number of records returned	Number of records (or array entries) returned.
BINARY(4)	Final select record	Record number (or array entry), within the set of records returned, of the Final Select ICON.
BINARY(4)	Reserved	Currently not used.
BINARY(4)	Reserved	Currently not used.

Type	Variable	Description
BINARY(4)	Reserved	Currently not used.
BINARY(4)	Reserved	Currently not used.
BINARY(4)	Reserved	Currently not used.

Usage Notes

This function is threadsafe, but not thread-enabled. Database monitor data is collected in the threaded process.

Output Format

The format for each record (or each array entry) in the output data is as follows:

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Unique ICON number. Each icon in the picture is given a unique number. This value is what ties the records together; that is, all records with the same unique ICON number are associated with that specific ICON.
4	4	BINARY(4)	Record type or ID. Each record can be one of many possible types. See “Record Types” on page 453 for the list of possible record types. A record type of 10 (new ICON) indicates another ICON was added to the picture and its unique number can be found in the unique ICON number field.
8	8	BINARY(4)	Context type. The type of context with which this record is associated. A context is a group of values used for a special purpose. See “Context Types” on page 452 for the list of possible contexts.
12	C	BINARY(4)	Context order. If this record is associated with a context type, this is the order, or position, within the context for the value. For example, table name would have a context order of two since it is the second variable of information associated with the table description context. Table library would have a context order of one.
16	10	BINARY(4)	Flyover order. The order, or position, within the flyover information for this value. The flyover information is the information that is shown when the cursor is held over a particular ICON and a window pops up showing some of the data attributes associated with that ICON.
20	14	BINARY(4)	Arrow order. The order, or position, within the arrow information for this value. The arrow information is the information shown on the arrows that connect the ICONs.
24	18	BINARY(4)	Arrow value. The value shown on the arrow that connects the ICONs together. Generally, this value is either the estimated number of rows or the estimated processing time.

Offset			
Dec	Hex	Type	Field
28	1C	BINARY(4)	Format value. Used to highlight or format output data. For example, all header lines within the data attributes will have a format value associated with them. This identifies all the header records and allows users the option to format these specific records in the same manner. See “Format Types” on page 453 for a list of format values.
32	20	CHAR(1)	Specifies (yes or no) if the data attributes associated with the ICON are returned in a predetermined (numerical) order. This variable is set only for the record whose context type is new ICON. This variable is pertinent only to those users who wish to show the data attributes in the same order that the Visual Explain consolidator returned them. Users who wish to choose their own order of data attributes can simply ignore this indicator.
33	21	CHAR(1)	Type of output data. The type of output data is either ″C″, ″N″, or ″X″. It is the field that contains the character output (″C″) or the field that contains the numeric output (″N″). It tells the user which field to look in for the output data. A value of ″X″ indicates the character output exceeds 1000 bytes and the remaining character output is found on the following array entry.
34	22	BINARY(2)	Length, in bytes, of the column heading.
36	24	CHAR(128)	Column heading. The description, or heading, of the output data. For a record type of new ICON, this will be the label of the ICON. For a data attribute record type, this will be a description of the data attribute; for example, Table name.
164	A4	BINARY(4)	Reserved.
168	A8	BINARY(4)	Reserved.
172	AC	BINARY(4)	Reserved.
176	B0	BINARY(2)	Length, in bytes, of the character output data. It is set only if the type of output is ″C″ or ″X″.
178	B2	CHAR(1000)	Character output. The character output could be many things, depending on the record type. For a new ICON record, the character output is the label associated with the ICON. For a data attribute, the character output is the value for that data attribute. For example, if the column heading was Table name, then the character output would be TABLE001. It is set only if the type of output is ″C″ or ″X″.
1178	49A	BINARY(4)	Numeric output. The numeric output is used only for a record that has a numeric output (that is, the child ICON record type or number of child ICONs record type). Generally, most records (especially data attribute records) have their output converted to character format. It is set only if the type of output is ″N″.
1182	49E	BINARY(2)	Reserved.

ICON Labels

To determine which ICON should be shown, look at the non-translated ICON label that is returned in the character output field. Compare this text string to the text string associated with the user-generated ICONs. The ICON labels that may be returned are shown below. For a detailed description of the operation represented by the ICON, refer to Database Performance and Query Optimization in the iSeries Information Center.

ICON Label	Description
Table Scan	All rows in the table were paged in and selection criteria was applied against each row. Only those rows meeting the selection criteria were retrieved. To get the result in a particular sequence, you must specify the ORDER BY clause.
Table Scan, Parallel	A table scan access method was used and multiple tasks were used to select the rows in parallel. The table was partitioned and each task was given a portion of the table with which to work.
Index Scan - Key Selection	All entries of the index were paged in. (This is different from key positioning, where only a specified range of key entries were paged in.) Any selection criteria, whose predicates match the key columns of the index, were applied against the index entries. Only selected key entries were used to select rows from the corresponding table data.
Index Scan - Key Selection, Parallel	Multiple tasks were used to perform key selection in parallel. The table was partitioned and each task was given a portion of the table with which to work.
Index Scan - Key Positioning	
Index Scan - Key Positioning, Parallel	Multiple tasks were used to perform the key positioning in parallel. The range of key values was determined by the selection criteria whose predicates matched the key columns of the index. The number of selected key entries were further reduced by the use of index key selection or derived selection after key positioning was completed. Only selected key entries were used to select rows from the corresponding table data.
Skip Sequential Table Scan	A bitmap was used to determine which rows would be selected. No CPU processing was done on rows not selected and I/O was minimized by bringing in only those pages that contained rows to be selected.
Skip Sequential Table Scan, Parallel	A skip sequential table scan access method was used and multiple tasks were used to select the rows in parallel. The table was partitioned and each task was given a portion of the table with which to work.
Encoded Vector Index	Access was provided to a database file by assigning codes to distinct key values, and then representing these values in an array (vector). The elements of the array can be 1, 2, or 4 bytes in length, depending on the number of distinct values that must be represented. Because of their compact size and relative simplicity, encoded vector indexes provide for faster scans that can be more easily processed in parallel.
Encoded Vector Index, Parallel	Multiple tasks were used to perform the encoded vector index selection in parallel. T his allows for faster scans that can be more easily processed in parallel. The elements of the array can be 1, 2, or 4 bytes in length, depending on the number of distinct values that must be represented. Because of their compact size and relative simplicity, encoded vector indexes provide for faster scans and can be more easily processed in parallel.

ICON Label	Description
Dynamic Bitmap	A bitmap was generated dynamically from an existing index. It then was used to determine which rows were to be retrieved from the table. To improve performance, dynamic bitmaps can be used in conjunction with any of the following access methods: • Skip sequential table scan • Index scan - key positioning • Index scan - key selection
Temporary Table	A temporary table was required to contain the intermediate results of the query, or the queried table could not be queried as it currently exists and a temporary table was created to replace it.
Temporary Hash Table	A temporary hash table was created to perform hash processing.
Temporary Index	A temporary hash table was created to perform hash processing.
Hash Join	A temporary hash table was created to perform the join. T he tables queried were joined together using a hash join implementation where a hash table was created for each secondary table. Therefore, matching values were hashed to the same hash table entry.
Nested Loop Join	Queried tables were joined together using a nested loop join implementation. Values from the primary file were joined to the secondary file using an index whose key columns matched the specified join columns.
Index Grouping	Selected rows were grouped or summarized. Therefore, duplicate rows within a group were eliminated.
Hash Grouping	Selected rows were grouped or summarized. Therefore, duplicate rows within a group were eliminated.
Sort	Selected rows were sorted using a sort algorithm.
Union Merge	The results of multiple subselects were merged or combined into a single result.
Subquery Merge	The results of multiple subselects were merged or combined into a single result.
Bitmap Merge	Multiple bitmaps were merged or combined to form a final bitmap. The merging of the bitmaps simulates boolean logic (AND/OR selection).
Distinct	Duplicate rows in the result were prevented. You can specify that you do not want any duplicates by using the DISTINCT keyword, followed by the selected column names.
Select	A point in the query where multiple results are brought together into a single result set. For example, if a query is the union of two different select statements, at the point before the union occurs, the Select icon indicates the points where the select statements finished and the union is about to occur.
Final Select	The original text and summary information of how the query was implemented.
Insert	The original text and summary information of how the query was implemented.
Update	The original text and summary information of how the query was implemented.
Delete	The original text and summary information of how the query was implemented.

ICON Label	Description
Unknown	The operation performed is not recognized by Visual Explain. For example, the system may support a new function that is not yet supported by Visual Explain.

Context Types

BINARY(4) A context is a group of values used for a special purpose.

Context	Type	Description
21	Table Description	The variables needed to retrieve information about the table. These variables are: 1. Table library 2. Table name
22	Index Description	The variables needed to retrieve information about the index. These variables are: 1. Index library 2. Index name
23	Create Index Attributes	The information needed to create an index. This includes: 1. Library of base table 2. Name of base table 3. Type of index to create *"B"* Binary radix index *"E"* Encoded vector index 4. Number of unique values 5. Key columns 6. Alternate collating sequence library name 7. Alternate collating sequence table name
24	Environment Attributes	Information about the environment when the query was executed. This includes: 1. Memory pool size 2. Memory pool ID 3. Date format 4. Date separator 5. Time format 6. Time separator 7. Decimal point 8. Sort sequence table name 9. Sort sequence library name 10. Language ID 11. Query INI table name 12. Query INI library name 13. Query time limit 14. Parallel degree 15. Maximum number of tasks 16. Parameter marker conversion

Format Types

BINARY(4) The formatting value is used to format or highlight similar output data. For example, all header lines within the data attribute output will have a format value associated with them. This allows the user the option to identify and format all these particular header lines in the same manner.

Format	Type	Description
8	Index Advised	Data attributes associated with the index advised function.
16	Header	Header line within the data attribute output.

Record Types

Generally, record types with a value less than 100 are used to construct the picture. For example, they determine which ICONs are connected together. Record types with a value greater than 1000 are data attributes (information associated with a particular ICON). For a detailed description of the data attributes, see Database Performance and Query Optimization in the iSeries Information Center.

Record Type	Description
10	New ICON.
11	Number of child ICONs.
12	Unique ICON number of the child ICON.
111	Heading only, no output data.
1010	Name of the index created.
1011	Library of the index created.
1012	Name of the temporary table created.
1013	Library of the temporary table created.
1014	Name of the temporary hash table created.
1015	Library of the temporary hash table created.
1031	Library of the table being queried.
1032	Name of the table being queried.
1033	Member name of the table being queried.
1034	Long name of the table being queried.
1035	Long library of the table being queried.
1041	Library of the base table.
1042	Name of the base table (underlying physical table).
1043	Member name of the base table.
1044	Long name of the base table.
1045	Long library of the base table.
1051	Library of the index used.
1052	Name of the index used.
1053	Member name of the index used.
1054	Long name of the index used.
1055	Long library of the index used.
1102	Time when the database monitor record was created.
1104	Timestamp of when the SQL statment started.
1106	Timestamp of when the SQL statement ended.
1108	Amount of time spent during optimization, in seconds.
1110	Amount of time spent creating the cursor (open data path), in seconds.
1112	Total time for the SQL statement, in milliseconds.
1114	Total time for the SQL statement, in microseconds.
1120	Statement OPEN time, in milliseconds.
1122	Statement FETCH time, in milliseconds.
1124	Statement CLOSE time, in milliseconds.
1220	Statement number.
1222	Statement function.
1224	Statement operation.

Record Type	Description
1226	Statement type.
1228	Statement name.
1230	Statement outcome.
1232	SQL return code.
1234	SQLSTATE.
1240	Cursor name.
1242	Package name.
1244	Package library.
1250	Number of rows returned.
1252	Number of rows fetched.
1260	SQL statement text.
1306	CLOSQLCSR value.
1308	ALWCPYDTA value.
1310	Pseudo open.
1312	Pseudo close.
1313	Hard close reason code.
1314	Open data path implementation.
1320	Dynamic replan reason code.
1324	Dynamic replan reason subcode.
1326	Timestamp of the last replan.
1330	Parse required.
1332	Data conversion.
1334	Level of commitment control.
1336	Blocking enabled.
1338	Delay preperation.
1339	Statement is explainable.
1340	Type of naming convention.
1342	Type of dynamic execution.
1344	Optimize LOB.
1350	User profile, static.
1352	User profile, dynamic.
1354	Default collection.
1360	Procedure name on the call.
1362	Procedure library on the call.
1364	Directory path.
2012	Estimated processing time, in seconds.
2016	Cumulative processing time, in seconds.
2018	Total number of rows in the table.
2020	Size of the table.
2042	Estimated number of rows selected.
2044	Estimated number of joined rows.
2046	Join position.
2048	Original file position.
2050	Join method.
2052	Join type.
2054	Join operator.
2056	Join fanout.
2058	Number of files joined.
2070	I/O or CPU bound.
2080	Reason code.
2110	Index scan, key positioning.
2112	Number of key columns for key positioning.
2114	Estimated number of entries selected through key positioning.
2116	Index scan, key selection.

Record Type	Description
2118	Estimated number of entries selected through key selection.
2122	Index only access.
2124	Index fits into main memory.
2126	Memory pool size.
2128	Memory pool ID.
2130	Skip key processing.
2140	Type of index.
2141	Index usage.
2142	Number of entries in the index.
2144	Number of unique values in the index.
2146	Percent overflow for the index.
2148	Vector size of the index.
2150	Size of the index used.
2152	Page size of the index used.
2154	Reason code of why index was used.
2160	Index is a constraint.
2162	Name of the constraint.
2182	Data space selection exists.
2184	Skip sequential processing was used.
2190	Reason code for the table scan processing.
2220	Index is a constraint.
2222	Name of the constraint.
2224	Data space selection exists.
2226	Skip sequential processing was used.
2320	The query optimizer timed out.
2322	Reason code of why the index was not used.
2324	List of indexes which the query optimizer considered.
2346	The query optimizer is advising an index to be created.
2348	The number of key columns within the index advised that will use key positioning.
2350	The list of key columns for the index advised.
2380	Was parallel pre-fetch used.
2382	Was parallel pre-load used.
2384	Parallel degree requested by the query optimizer.
2386	Parallel degree used.
2388	Reason code why the parallel degree requested by the optimizer was not used.
2402	Number of entries in the temporary index created.
2404	Page size of the temporary index created.
2406	Row size of the temporary index created.
2408	Was an alternate collating sequence table used to create the temporary index.
2409	Name of the alternate collating sequence table used to create the temporary index.
2410	Library of the alternate collating sequence table used to create the temporary index.
2412	Is the temporary index that was created reusable.
2414	Is the temporary index that was created a sparse or select/omit index.
2416	Type of index that was created.
2418	Was the index created as a permanent object.
2420	Was the index created from another index.
2422	Parallel degree requested by query optimizer for creation of the index.
2424	Parallel degree used during creation of the index.
2426	Reason code why the parallel degree requested by the optimizer for the index creation was not used.
2428	Reason code why a temporary index was created.
2430	Key columns used when creating the temporary index.
2510	Number of rows within the temporary table.
2512	Size of the temporary table.

Record Type	Description
2514	Row size of the temporary table.
2516	Default values exist in temporary table.
2518	Temporary table created is a temporary result table.
2520	Temporary table created is a distributed table.
2522	Nodes where the temporary distributed table was created.
2524	Reason code why a temporary table was created.
2550	Number of rows within the temporary hash table.
2552	Size of the temporary hash table.
2554	Row size of the temporary hash table.
2556	Key size of the temporary hash table.
2558	Element size of the temporary hash table.
2560	Memory pool size where temporary hash table was created.
2562	Memory pool ID where temporary hash table was created.
2563	Reason code why a temporary hash table was created.
2564	Columns used when creating the temporary hash table.
2612	Columns used for dataspace selection.
2614	Was derived selection used.
2616	Columns used for derived selection.
2620	Columns used for key positioning.
2622	Columns used for key selection.
2624	Columns used for join selection.
2626	Columns used for ordering.
2628	Columns used for grouping.
2810	Type of grouping implementation.
2812	Does HAVING selection exist.
2814	Was the HAVING selection converted into WHERE seletion.
2816	Estimated number of groups.
2818	Average number of rows within each group.
2820	Grouping columns.
2822	MIN columns.
2824	MAX columns.
2826	SUM columns.
2828	COUNT columns.
2830	AVERAGE columns.
2910	Subselect number of the inner subselect.
2912	Nested level of the inner subselect.
2914	Subselect number of the materialized view containing the inner subselect.
2916	Nested level of the materialized view containing the inner subselect.
2920	Subquery operator.
2922	Correlated columns exist.
2924	List of the correlated columns.
3020	Size of the bitmap created.
3022	Number of bitmaps created.
3024	IDs of the bitmaps created.
3026	IDs of the bitmaps that were merged together.
4020	System name.
4022	Job name.
4024	Job user.
4026	Job number.
4028	Unique query count.
4032	Subselect count.
4040	Relational database name.
4042	Thread ID.
4044	Unique refresh count.

Record Type	Description
4046	Subselect nested level.
4048	Materialization number of the subselect.
4050	Nested level of the subselect that was materialized.
4052	Materialization number for the decomposed subselect.
7008	List of the host variable values.
7009	Type of host variable implementation.
7010	Type of processing for the specified ordering.
7011	Name of the index used to satisfy ordering.
7012	Library of the index used to satisfy ordering.
7013	Long name of the index used to satisfy ordering.
7014	Long library of the index used to satisfy ordering.
7020	Type of processing for the specified grouping.
7021	Name of the index used to satisfy grouping.
7022	Library of the index used to satisfy grouping.
7023	Long name of the index used to satisfy grouping.
7024	Long library of the index used to satisfy grouping.
7026	Query contains UNION.
7027	Query contains subquery (subselect).
7030	Type of join processing.
7032	Query contains distinct.
7034	Query contains distributed tables.
7036	List of the nodes containing the distributed tables.
7050	Quick summary of the implementation.
8014	Memory pool size.
8016	Memory pool ID.
8020	Date format.
8022	Date separator.
8024	Time format.
8026	Time separator.
8028	Decimal point.
8030	Name of the sort sequence table associated with the query.
8032	Library of the sort sequence table associated with the query.
8034	Language ID.
8036	Country or region ID.
8040	Query INI table name.
8042	Query INI library.
8044	Maximum query time limit.
8046	Parallel options.
8048	Maximum number of tasks.
8050	Apply CHGQRYA options to remote systems.
8052	Asynchronous job usage.
8054	Join order was forced.
8056	Print debug messages.
8060	Parameter marker conversion.
8062	User defined function (UDF) time limit.
8064	Optimizer limitations.

Error Codes

Possible error codes returned from the Visual Explain consolidator are:

Error Code	Description
0	Successful.
71	Invalid date format.

Error Code	Description
72	Invalid date separator.
73	Invalid time format.
74	Invalid time separator.
75	Invalid decimal point.
90	No records in the specified database monitor table.
91	Failure trying to read records from specified database monitor table.
92	Query too complex to be explained.
93	Specified database monitor table not found.
99	Query function not supported.
1000	Missing or invalid QQQ1000 record within the database monitor table.
3000	Missing or invalid QQQ3000 record within the database monitor table.
3001	Missing or invalid QQQ3001 record within the database monitor table.
3002	Missing or invalid QQQ3002 record within the database monitor table.
3003	Missing or invalid QQQ3003 record within the database monitor table.
3004	Missing or invalid QQQ3004 record within the database monitor table.
3014	Missing or invalid QQQ3014 record within the database monitor table.
3021	Missing or invalid QQQ3021 record within the database monitor table.
3022	Missing or invalid QQQ3022 record within the database monitor table.
3023	Missing or invalid QQQ3023 record within the database monitor table.
3025	Missing or invalid QQQ3025 record within the database monitor table.
3027	Missing or invalid QQQ3027 record within the database monitor table.
3028	Missing or invalid QQQ3028 record within the database monitor table.
0nnn	SQL error code (converted to a positive value) that occurred while reading records from the specified database monitor table.

API introduced: V5R1

Top | "Database and File APIs," on page 1 | APIs by category

Structured Query Language (SQL)APIs

The SQL APIs are:

- Add or replace labels in catalog descriptions (LABEL) adds or replaces labels in the catalog descriptions of tables, views, aliases, packages, or columns.
- Allow LOB locator to retain association with value (HOLD LOCATOR) allows a LOB locator variable to retain its association with a value beyond a unit of work.
- Alter a table (ALTER TABLE) alters the description of a table.
- Assign value to CURRENT PATH special register (SET PATH) changes the value of the CURRENT PATH special register.
- Assign value to CURRENT SCHEMA special register (SET SCHEMA) changes the value of the CURRENT SCHEMA special register.
- Assign value to parameter or variable (assignment-statement) assigns a value to an SQL parameter or SQL variable.
- Assign values to a host variable (SET variable) produces a result table consisting of at most one row and assigns the values in that row to host variables.
- Assign values to a transition variable (SET transition-variable) assigns values to a transition variable.
- Assign values to host variables (SELECT INTO) produces a result table consisting of at most one row, and assigns the values in that row to host variables.
- Branch to user-defined label (GOTO) branches to a user-defined label within an SQL routine or SQL trigger.
- Call a procedure (CALL) calls a procedure.

- Cause flow of control to return to loop (ITERATE) causes the flow of control to return to the beginning of a labelled loop.
- "Change Dynamic Default Collection (QSQCHGDC) API" on page 462 (QSQCHGDC) defines a default collection for unqualified table names in dynamically prepared statements or in dynamically executed statements.
- Change isolation level for unit of work (SET TRANSACTION) sets the isolation level and read only attribute for the current unit of work.
- Close a cursor (CLOSE) closes a cursor.
- Comment on various objects (COMMENT) replaces or adds a comment to the description of an alias, column, function, index, package, parameter, procedure, table, type or view.
- Connect to a server and establish rules (Type 2) (CONNECT) connects an activation group within an application process to the identified server using the rules for application directed distributed unit of work.
- Connect to server and establish rules (Type 1) (CONNECT) connects an activation group within an application process to the identified server using the rules for remote unit of work.
- Continue execution (LEAVE) continues execution by leaving a block or loop.
- Create a distinct type (CREATE DISTINCT TYPE) defines a distinct type at the current server.
- Create a function based on another existing function (CREATE FUNCTION) creates a user-defined function, based on another existing scalar or column function, at the current server.
- Create a procedure (CREATE PROCEDURE) defines a procedure at the current server.
- Create a schema and objects in that schema (CREATE SCHEMA) defines a schema at the current server and optionally creates tables, views, aliases, indexes, and distinct types.
- Create a table (CREATE TABLE) defines a table at the current server.
- Create a trigger (CREATE TRIGGER) defines a trigger at the current server.
- Create a user-defined function (CREATE FUNCTION) defines a user-defined function at the current server.
- Create an alias (CREATE ALIAS) defines an alias on a table, view, or member of a database file at the current server.
- Create an external procedure (CREATE PROCEDURE) defines an external procedure at the current server.
- Create an external scalar function (CREATE FUNCTION) creates an external scalar function at the current server.
- Create an external table function (CREATE FUNCTION) creates an external table function at the current server.
- Create an index on a table (CREATE INDEX) creates an index on a table at the current server.
- Create an SQL procedure (CREATE PROCEDURE) creates an SQL procedure at the current server.
- Create an SQL scalar function (CREATE FUNCTION) creates an SQL scalar function at the current server.
- Create an SQL table function (CREATE FUNCTION) creates an SQL table function at the current server.
- Create view of table (CREATE VIEW) creates a view on one or more tables or views at the current server.
- Declare names identifying SQL statements (DECLARE STATEMENT) is used for program documentation and declares names that are used to identify prepared SQL statements.
- Declare subtype or CCSID (DECLARE VARIABLE) is used to assign a subtype or CCSID other than the default to a host variable.
- Define a declared global temporary table (DECLARE GLOBAL TEMPORARY TABLE) defines a declared temporary table for the current application process.
- Define actions to take on SQL return codes (WHENEVER) specifies the action to be taken when a specified exception condition occurs.

- Define an external procedure (DECLARE PROCEDURE) defines an external procedure.
- Define an SQL cursor (DECLARE CURSOR) defines an SQL cursor.
- Delete rows from a table (DELETE) deletes rows from a table or view.
- Describe result columns (DESCRIBE) obtains information about a prepared statement.
- Drop an object (DROP) drops an alias, function, index, package, procedure, schema, table, trigger, type, or view.
- End a unit of work (COMMIT) ends a unit of work and commits the database changes made by that unit of work.
- End a unit of work (ROLLBACK) ends a unit of work and back out all the relational database changes, or back out only the changes made after a savepoint was set.
- End connection (DISCONNECT) ends one or more connections for unprotected conversations.
- Establish options for processing SQL statements (SET OPTION) establishes the processing options to be used for SQL statements.
- Execute a query (SELECT) executes a query.
- Execute prepared SQL statement (EXECUTE) executes a prepared SQL statement.
- Execute statement for rows of a table (FOR) executes a statement for each row of a table.
- "Generate Data Definition Language (QSQGNDDL) API" on page 463 (QSQGNDDL) generates the SQL data definition language statements required to recreate a database object.
- Grant privilege on a distinct type (GRANT) grants privileges on a distinct type.
- Grant privilege on a function or procedure (GRANT) grants privileges on a function or procedure.
- Grant privilege on a package (GRANT) grants privileges on a package.
- Grant privileges on a table or view (GRANT) grants privileges on tables or views.
- Group statements in SQL routine (compound-statement) groups other statements together in an SQL procedure.
- Identify result sets in a procedure (SET RESULT SET) identifies one or more result sets that can be returned from an external procedure when the procedure is called by a iSeries Access client, the SQL Call Level Interface, or when accessed from a remote system using DRDA.
- Insert declarations into source program (INCLUDE) inserts declarations or statements into a source program.
- Insert rows into a table (INSERT) inserts rows into a table or view.
- Mark beginning of SQL declare section (BEGIN DECLARE SECTION) marks the beginning of an SQL declare section.
- Mark end of SQL declare section (END DECLARE SECTION) marks the end of an SQL declare section.
- Obtain information about a table (DESCRIBE TABLE) obtains information about a table or view.
- Obtain information about SQL statement (GET DIAGNOSTICS) obtains information about the previous SQL statement that was executed.
- Open a cursor (OPEN) opens a cursor.
- Place one or more connections in the release-pending state. (RELEASE (Connection)) places one or more connections in the release-pending state.
- Position cursor on table (FETCH) positions a cursor on a row of the result table, and can also assign values from one or more rows of the result table to host variables.
- Prepare an SQL statement for execution (PREPARE) creates an executable form of an SQL statement from a character-string form of the statement.
- Prepare and execute an SQL statement (EXECUTE IMMEDIATE) prepares and executes an SQL statement.
- Prevent changing or using a table (LOCK TABLE) prevents either concurrent processes from changing a table or prevents concurrent processes from using a table.

- “Process Extended Dynamic SQL (QSQPRCED) API” on page 476 (QSQPRCED) processes Structured Query Language (SQL) extended dynamic statements in an SQL package object.
- Provide conditional execution (IF) executes different sets of SQL statements based on the result of search conditions.
- Provide method to invoke function (VALUES) provides a method to invoke a user-defined function from a trigger.
- Release savepoint within unit of work (RELEASE SAVEPOINT) releases the identified savepoint and any subsequently established savepoints within a unit of work.
- Remove association of LOB locator and its value (FREE LOCATOR) removes the association between a LOB locator variable and its value.
- Rename a table, view, or index (RENAME) renames a table, view, or index.
- Repeat execution of a statement (REPEAT) executes a statement or group of statements until a search condition is true.
- Repeat execution of a statement (LOOP) repeats the execution of a statement or a group of statements.
- Repeat execution of statement (WHILE) repeats the execution of a statement while a specified condition is true.
- Resignal an error or warning condition (RESIGNAL) is used within a handler to return an error or warning condition.
- Return from a routine (RETURN) returns from a routine.
- Revoke distinct type privileges (REVOKE) removes the privileges on a distinct type.
- Revoke function or procedure privileges (REVOKE) removes the privileges on a function or procedure.
- Revoke package privileges (REVOKE) removes the privilege to execute statements in a package.
- Revoke table privileges (REVOKE) removes privileges on a table or view.
- Select path (CASE) selects an execution path based on multiple conditions.
- Set a savepoint within unit of work (SAVEPOINT) sets a savepoint within a unit of work to identify a point in time within the unit of work to which relational database changes can be rolled back.
- Set connection to establish server (SET CONNECTION) establishes the current server of the activation group by identifying one of its existing connections.
- Signals an error or warning condition (SIGNAL) causes an error or warning to be returned with the specified SQLSTATE and optional message text.
- Specify a result table (VALUES INTO) produces a result table consisting of at most one row and assigns the values in that row to host variables.
- » “sqludf_append()—SQL LOB Append to Locator” on page 502 (sqludf_append()) appends data to the end of the LOB data the locator represents.«
- » “sqludf_create_locator()—SQL LOB Create Locator” on page 505 (sqludf_create_locator()) creates a LOB locator.«
- » “sqludf_create_locator_with_ccsid()—SQL LOB Create Locator With CCSID” on page 509 (sqludf_create_locator_with_ccsid()) creates a LOB locator with a given CCSID.«
- » “sqludf_free_locator()—SQL LOB Free Locator” on page 513 (sqludf_free_locator()) frees a LOB locator.«
- » “sqludf_length()—SQL LOB locator length” on page 515 (sqludf_length()) returns the length of the LOB data represented by a LOB locator.«
- » “sqludf_substr()—SQL LOB Substring Locator” on page 519 (sqludf_substr()) returns a substring of the LOB data the locator represents.«
- “Syntax Check SQL Statement (QSQCHKS) API” on page 523 (QSQCHKS) calls the DB2 for iSeries SQL parser to check the syntax of an SQL statement.
- Update values of columns in rows of a table (UPDATE) updates the values of specified columns in rows of a table or view.

«

Change Dynamic Default Collection (QSQCHGDC) API

Required Parameter Group:

1	Default collection name	
Input	Char(18)	

Optional or Omissible Parameter:

2	Error code	
I/O	Char(*)	

Default Public Authority: *USE

Threadsafe: Yes

The Change Dynamic Default Collection (QSQCHGDC) API defines a default collection for unqualified table names in dynamically prepared statements or in dynamically executed statements. The default collection is defined only for the job issuing the API call. The default collection will take precedence over the naming convention and default collection specified when the SQL program was created.

Authorities and Locks

None.

Required Parameter

Default collection name
INPUT; CHAR(18)

The name of the default collection. The following values are allowed:

**CURLIB*	The current library at the time the API is called is used as the default collection. Subsequent changes of the current library will not change the default collection. If no current library is defined, library QGPL is used.
**PGM*	The default collection is determined by the attributes specified when the SQL program was created. If DYNDFTCOL(*YES) was specified, the default collection is the library name specified on the DFTRDBCOL keyword. If DYNDFTCOL(*NO) was specified, dynamically prepared and executed statements will use the default collection rules based on the naming convention. For further information on naming conventions, see the DB2 Universal Database for iSeries SQL Reference.
default collection name	The name of the default collection. This value must be uppercase and not delimited.

The API does not validate the existence or the user's authority to the specified default collection. These validations occur on the execution of subsequent SQL prepare or execute operations.

Optional or Omissible Parameter

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error code parameter. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Usage Notes

1. If a package is created with a DFTRDBCOL, QSQCHGDC has no effect.
2. In V5R2, the scope of QSQCHGDC was changed from job scoped to activation group scoped.

Error Messages

Message ID	Error Message Text
CPF3CF1 E	Error code parameter not valid.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

API introduced: V4R5

Top | "Database and File APIs," on page 1 | APIs by category

Generate Data Definition Language (QSQGNDDL) API

Required Parameter Group:

1	Input template
Input	Char(*)
2	Length of input template
Input	Binary(4)
3	Input template format name
Input	Char(8)
4	Error code
I/O	Char(*)

Default Public Authority: *USE

Threadsafe: No

The Generate Data Definition Language (QSQGNDDL) API generates the SQL data definition language statements required to recreate a database object. The results are returned in the specified database source file member.

Database physical files or logical files that were created using an interface other than SQL may be specified. For example, files created from DDS and the CRTPF or CRTLF commands may be specified. Even if the object was created using SQL, the Standards option may restrict what can be generated. In either of these cases:

- the operation may succeed with warnings that are generated in the SQL statement source, or
- the operation may fail for certain non-relational files or objects not supported by the specified Standards option.

If a database object was created using an SQL interface, the resulting SQL statements may be slightly different than the SQL statements that created the object originally. For example:

- When there is more than one way to specify an attribute in SQL, the more standard syntax is generally chosen. For example, if a user creates a table with a FLOAT(52) column, DOUBLE PRECISION is generated.
- When a clause is not specified in the original SQL statement and a default is taken instead, a clause may be generated to explicitly show the default. For example, if the default value for a nullable column is the null value, the clause DEFAULT NULL is generated.
- When a Standards option is used to restrict the generated SQL to the ANS and ISO standard or the DB2 Universal Database Family, an attribute may be omitted. For example, if the ALLOCATE clause is specified on a VARCHAR column, the ALLOCATE clause is not generated unless the Standards option allows DB2 UDB for iSeries extensions.

For more information, see the Severity level field within the “SQLR0100 Format” on page 465.

You can use the QSQGNDDL API with database objects only. DDM files (other than SQL aliases) are not supported. File overrides do not affect the specified object names. File overrides do affect the specified source file names.

Authorities and Locks

Object Library Authority
*EXECUTE

Source File Library Authority
*EXECUTE

Object Authorities
*EXECUTE for *LIB objects.
*USE for the *DTADCT object in a library (if SCHEMA is specified for the object type).
*USE for *FILE objects (not including aliases).
*USE to QSYS2/SYSPARMS for functions and procedures.
*USE to QSYS2/SYSROUTINE for functions and procedures.
*USE to QSYS2/SYSTYPES for types.
*USE to QSYS2/SYSTABLES for aliases.

Source File Authority
*OPER and *ADD.
If replace is specified, *DLT and either *OBJMGT or *OBJALTER is required also.

Object Lock
*SHRRD for *LIB objects.
*SHRNUP for *FILE objects (not including aliases). (See note below.)
*SHRRD to QSYS2/SYSFUNCS for functions.
*SHRRD to QSYS2/SYSPARMS for functions and procedures.
*SHRRD to QSYS2/SYSPROCS for procedures.
*SHRRD to QSYS2/SYSTYPES for types.
*SHRRD to QSYS2/SYSTABLES for aliases.

Note: If the object is a *FILE object, the lock is acquired only on the file definition and not the data. Applications that modify data can run concurrently with this API.

Source File Lock
*EXCLRD.

Required Parameter Group

Input template
INPUT;CHAR(*)

A structure that contains the input options used to generate DDL for the requested database object. For the format of this parameter, see "SQLR0100 Format."

Length of input template
INPUT; BINARY(4)

A variable that contains the length of the input template. The length must be greater than zero and large enough to contain all the template fields up to and including the Header Option. The length must not be larger than 32767.

Input template format name
INPUT; CHAR(8)

The format of the input template being used. The possible value is:

SQLR0100	Basic template

For more information, see "SQLR0100 Format."

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter. If this parameter is omitted, diagnostic and escape messages are issued to the application.

SQLR0100 Format

The following table shows the format of the input template parameter for the SQLR0100 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 466.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(258)	Database object name
258	102	CHAR(258)	Database object library name
516	204	CHAR(10)	Database object type
526	20E	CHAR(10)	Database source file name
536	218	CHAR(10)	Database source file library name
546	222	CHAR(10)	Database source file member name
556	22C	BINARY(4)	Severity level
560	230	CHAR(1)	Replace option
561	231	CHAR(1)	Statement formatting option
562	232	CHAR(3)	Date format
565	235	CHAR(1)	Date separator
566	236	CHAR(3)	Time format
569	239	CHAR(1)	Time separator
570	23A	CHAR(3)	Naming option
573	23D	CHAR(1)	Decimal point
574	23E	CHAR(1)	Standards option
575	23F	CHAR(1)	Drop option
576	240	BINARY(4)	Message level
580	244	CHAR(1)	Comment option

Offset		Type	Field
Dec	Hex		
581	245	CHAR(1)	Label option
582	246	CHAR(1)	Header option
583	247	CHAR(*)	Reserved

Field Descriptions

Comment option. The comment option specifies whether COMMENT ON SQL statements should be generated if a comment exists on the specified database object. If comments are not supported by the specified database object, the comment option is ignored. The valid values are:

0	COMMENT ON SQL statements should not be generated.
1	COMMENT ON SQL statements should be generated. If the specified database object type is a table or view, COMMENT ON SQL statements will also be generated for columns of the table or view.

If the Standards option is ′2′, comment option ′1′ is not valid.

Database object name. The name of the database object for which DDL will be generated. Either the SQL name or the system name may be specified. The name is case sensitive. If delimiters are required for the name to be valid, they must be specified. For example, a file with a name of ″abc″ must be specified with the surrounding quotes. A file with a name of ABC must be specified in upper case.

If the object type is a FUNCTION or PROCEDURE, this name must be the specific name of the function or procedure.

If TABLE or VIEW is specified for the object type, the object name may identify an alias. In this case, the object that the alias points to will be generated. A CREATE ALIAS statement will be generated only if ALIAS is specified for the object type.

Database object library name. The name of the library containing the object for which DDL will be generated. The name is case sensitive. If delimiters are required for the name to be valid, they must be specified. This name is ignored if the specified object type is SCHEMA. You can use these special values for the library name:

**CURLIB*	The job's current library
**LIBL*	The library list

Database object type. The type of the database object or object attribute for which DDL is generated. You can use these special values for the object type:

ALIAS	The object is an SQL alias. If the Standards option is ′2′, an ALIAS object type is not valid.
CONSTRAINT	The object attribute is a constraint.
FUNCTION	The object is an SQL function.
INDEX	The object is an SQL index. If the Standards option is ′2′, an INDEX object type is not valid.
PROCEDURE	The object is an SQL procedure.
SCHEMA	The object is an SQL schema (collection) or library.
TABLE	The object is an SQL table or physical file.
TRIGGER	The object attribute is a trigger.
TYPE	The object is an SQL type.

VIEW	The object is an SQL view or logical file.

Database source file name. The name of the source file that contains the SQL statements generated by the API. The name must be a valid system name. The name is case sensitive. If delimiters are required for the name to be valid, they must be specified. For example, a file with a name of ″abc″ must be specified with the surrounding quotes. A file with a name of ABC must be specified in upper case.

The record length of the specified source file must be greater than or equal to 92.

Database source file library name. The name of the library containing the source file that contains the SQL statements generated by the API. The name must be a valid system name. The name is case sensitive. If delimiters are required for the name to be valid, they must be specified. You can use these special values for the library name:

**CURLIB*	The job's current library
**LIBL*	The library list

Database source file member name. The name of the source file member that contains the SQL statements generated by the API. The name must be a valid system name. The name is case sensitive. If delimiters are required for the name to be valid, they must be specified. You can use these special values for the member name.

**FIRST*	The first database physical file member found.
**LAST*	The last database physical file member found.

Date format. The date format used for date constants in a generated SQL CREATE TABLE statement. The date format may not apply to date constants that are in ISO, EUR, USA, or JIS format in a CREATE VIEW, CREATE TRIGGER, CREATE FUNCTION, or CREATE PROCEDURE statement. The valid values are:

ISO	International Standards Organization (yyyy-mm-dd)
EUR	IBM European Standard (dd.mm.yyyy) If the Standards option is ′2′, the EUR date format is not valid.
JIS	Japanese Industrial standard Christian Era (yyyy-mm-dd) If the Standards option is ′2′, the JIS date format is not valid.
USA	IBM USA standard (mm/dd/yyyy) If the Standards option is ′2′, the USA date format is not valid.
MDY	Month/day/year (mm/dd/yy) If the Standards option is ′1′ or ′2′, the MDY date format is not valid.
DMY	Day/month/year (dd/mm/yy) If the Standards option is ′1′ or ′2′, the DMY date format is not valid.
YMD	Year/month/day (yy/mm/dd) If the Standards option is ′1′ or ′2′, the YMD date format is not valid.
JUL	Julian (yy/ddd) If the Standards option is ′1′ or ′2′, the JUL date format is not valid.

Date separator. The date separator used for date constants in a generated SQL CREATE TABLE statement. The date separator may not apply to date constants that are in ISO, EUR, USA, or JIS format in a CREATE VIEW, CREATE TRIGGER, CREATE FUNCTION, or CREATE PROCEDURE statement. The date separator is only applicable if the date format is MDY, DMY, YMD, or JUL. The valid values are:

/	Slash separator
.	Period separator
,	Comma separator

-	Dash separator
blank	Blank separator

Decimal point. The decimal point used for numeric constants. The valid values are:

.	Period separator
,	Comma separator If the Standards option is ’1’ or ’2’, the comma separator is not valid.

Drop option. The drop option specifies whether DROP (or ALTER) SQL statements should be generated prior to the CREATE statement to drop the specified object. The valid values are:

0	DROP statements should not be generated.
1	DROP statements should be generated.

Note that with the exception of DROP SCHEMA, the DROP statements generated will not include a CASCADE or RESTRICT option even if the standards option is ’2’.

Header option. The header option specifies whether a header should be generated prior to the CREATE statement. The header consists of comments that describe the version, date and time, the relational database, and some of the options used to generate the SQL statements. The valid values are:

0	A header should not be generated.
1	A header should be generated.

Label option. The label option specifies whether LABEL ON SQL statements should be generated if a label exists on the specified database object. If labels are not supported by the specified database object, the label option is ignored. The valid values are:

0	LABEL ON SQL statements should not be generated.
1	LABEL ON SQL statements should be generated. If the specified database object type is a table or view, LABEL ON SQL statements will also be generated for columns of the table or view.

If the Standards option is ’1’ or ’2’, label option ’1’ is not valid.

Message level. The severity level at which the messages are generated. If errors occur that have a severity level greater than this value, a message is generated in the output. The valid values are in the range 0 through 39 inclusive.

The message level must be less than or equal to the severity level.

Naming option. The naming convention used for qualified names in the generated SQL statements. The valid values are:

SQL	*collection.table* syntax
SYS	*library/file* syntax

If the Standards option is ’1’ or ’2’, the SYS naming option is not valid.

If the object type is a FUNCTION, PROCEDURE, TRIGGER, or VIEW, and a column name is qualified by a qualified table name in the SQL body of the function, procedure, trigger, or view (that is, schema-name.table-name.column-name), the generated statement will not be valid because this type of column name qualification is not allowed in SYS naming.

Replace option. The replace option for the database source file member. The valid values are:

0	The resulting SQL statements are appended to the end of the database source file member.
1	The database source file member is cleared prior to adding the resulting SQL statements. If this option is chosen, the file may be cleared even if an error is returned from the API.

Reserved. A reserved field. It must contain hexadecimal zeroes.

Severity level. The severity level at which the operation fails. If errors occur that have a severity level greater than this value, the operation ends. The valid values are in the range 0 through 39 inclusive. Any severity 40 error will cause the API to fail.

0	No errors or warnings.

10 The following attributes will result in messages with this severity level:

- Schema ASP and WITH DATA DICTIONARY

  If the Standards option is 1 or 2, these clauses will be ignored.
- Test libraries

  A CREATE SCHEMA statement will be generated to create the schema. Schemas are production libraries.
- Libraries with a CRTAUT parameter value

  Under SQL naming, schemas are always created with CRTAUT(*EXCLUDE). Under SYS naming, schemas are always created with CRTAUT(*SYSVAL).
- NODEGROUPs

  If the Standards option is 1 or 2, the NODEGROUP clause will beignored.
- LABEL ON TEXT

  If the Standards option is 1, the text will be ignored.
- COMMENT ON parameters

  If the Standards option is 1, the comment will be ignored.
- System file names

  If the Standards option is 1 or 2, only the SQL names are generated. Otherwise, a RENAME statement is generated after the CREATE statement to assign the system name.
- System column names

  If the Standards option is 1 or 2, only the SQL names are generated. Otherwise, a FOR COLUMN clause will be generated to assign each system column name.
- BIGINT data types

  If the Standards option is 1 or 2, a DECIMAL(19,0) will be generated.
- DBCS-open data types

  If the Standards option is 1 or 2, a character field will be generated.
-
  - Binary with non-zero scale

    A decimal data type will be generated.
  - Files whose format name is different from the file name

    The format name will be the same as the file name.
  - Files with a REUSEDLT(*NO) attribute

    REUSEDLT(*YES) will be used.
  - Physical or logical files that use any of the following keywords: CHECK, CHKMSGID, CMP, DATFMT, EDTCDE, EDTWRD, TIMFMT, RANGE, REFSHIFT, VALUES

    These keywords will be ignored.
  - Logical files that use any of the following keywords: CCSID or TRNTBL

    These keywords will be ignored.
  - Join logical files with JDFTVAL or JDUPSEQ

    A LEFT OUTER JOIN clause will be generated, but the join default value will be the null value and the JDUPSEQ keyword will be ignored.
  - Logical files with SST function

    If the Standards option is 2, SUBSTRING is generated instead of SUBSTR.
  - COBOLLE and C++ languages in external functions and procedures

    If the Standards option is 1 or 2, COBOL or C is generated.
  - RPGLE language in external functions and procedures

    If the Standards option is 1, RPG is generated.

20 The following attributes will result in messages with this severity level:

- Multiple member files, files with no members, or files with MAXMBRS greater than one

  The resulting file will contain one member.
- Single format logical files with a member built over multiple physical file members

  The resulting file will be based on the first physical file member.
- Logical files that contain input/output fields that map an underlying physical file field to a different data type, length, precision or scale.

  A CAST scalar function will be generated to map the data to the correct attributes, but the resulting column is input-only.
- Keyed logical files that do not share the based on physical file's format, have more than one based on file, or have select/omit specifications

  If INDEX is specified, the format and select/omit will be ignored.
- Triggers with MODE DB2ROW

  If the Standards option is 1 or 2, MODE DB2SQL will be used.

30 The following attributes will result in messages with this severity level:

- CHAR or VARCHAR CCSID 65535

  If the Standards option is 2, a character field is generated.
- GRAPHIC, VARGRAPHIC, or DBCLOB

  If the Standards option is 2, a character field is generated.
- DataLinks or Row IDs

  If the Standards option is 1 or 2, a character field is generated.
- Identity columns

  If the Standards option is 2, the IDENTITY attribute is ignored.
- Open, Only, or Either fields

  If the Standards option is 0, the CCSID clause will result in an open field. Only and Either fields will result in a warning. If the Standards option is 1, FOR MIXED DATA is generated. If the Standards option is 2, character fields will be generated.
- Keyed logical files

  If VIEW is specified, the key specifications will be ignored, because all views are non-keyed.
- Keyed physical files whose key is not a primary key

  A CREATE TABLE will be generated without a primary key. The key specifications will be ignored, however, because only tables with a primary key are keyed.
- Files that use any of the following keywords: ALTSEQ, DIGIT, FCFO, FIFO, LIFO, UNSIGNED, ZONE

  These keywords will be ignored.
- SRTSEQ

  The sort sequence will be ignored.
- Non-SQL triggers if TABLE object is specified.

  The triggers will be ignored.
- NO EXTERNAL ACTION, SCRATCHPAD, FINAL CALL, ALLOW PARALLEL, or DBINFO, keywords in functions and procedures

  If the standards option is 2, these attributes will be ignored.
- COMMIT ON RETURN YES, NOT FENCED, or NEW SAVEPOINT LEVEL clauses in functions and procedures

  If the standards option is 1 or 2, these attributes will be ignored.
- Functions and procedures with parameter style GENERAL WITH NULLS, DB2SQL, or DB2GENERAL

  If the Standards option is 2, PARAMETER STYLE SQL is used.
- JAVA, REXX, RPG, and RPGLE language in functions and procedures

  If the Standards option is 2, the C language is used instead.
- CL language in functions and procedures

  If the Standards option is 1 or 2, the C language is used instead.

40 The following attributes will result in messages with this severity level:
- Physical file if either VIEW or INDEX object type is specified.
- Logical file if TABLE object type is specified.
- Non-keyed file if INDEX object type is specified.
- Non-alias file if ALIAS object type is specified.
- Function if PROCEDURE object type is specified.
- Procedure if FUNCTION object type is specified.
- Device files
- Program described physical files
- Multiple format logical files
- Indexes if the Standards option is 2.
- Aliases if the Standards option is 2.
- EVI Indexes if the Standards option is 1.
- UNIQUE WHERE NOT NULL if the Standards option is 1.
- Aliases that contain a member name if the Standards option is 1.
- System-generated UDFs
- Built-in data types
- SQL UDFs, if the Standards option is 1.
- Sourced UDFs, if the Standards option is 2.
- User-defined table functions, if the Standards option is 2.
- Non-SQL triggers if TRIGGER object is specified.

Standards option. The standards option specifies whether the generated SQL statements should contain DB2 UDB for iSeries extensions or whether the statements should conform to the DB2 Universal Database Family SQL or to the ANS and ISO SQL standards. The valid values are:

0 DB2 Universal Database for iSeries extensions may be generated in SQL statements.

1 The generated SQL statements must conform to SQL statements common to the DB2 Universal Database Family.

2 The generated SQL statements must conform to the following ANSI and ISO SQL standards:
- ISO (International Standards Organization) 9075-1: 1999, Database Language SQL
- ANSI (American National Standards Institute) X3.135-1-1999, Database Language SQL

If option 1 or 2 is chosen, the SQL statements generated may not completely represent the object in DB2 UDB for iSeries; however, the statements will be compatible with the specified DB2 Family or ANSI and ISO standards option.

If the object is an SQL function, SQL procedure, SQL trigger, or SQL view, the SQL statements in the body of the object are included in the generated SQL statement. Hence, if the option 1 or 2 is chosen, the generated SQL statement may not conform to the specified standards option since the statements within the body of the SQL object may not conform to the specified standard. For example, if a CREATE INDEX statement exists in the body of an SQL procedure, the generated CREATE PROCEDURE statement will contain the CREATE INDEX statement even if option 1 or 2 is chosen.

There is no attempt to take product specific limits into account. For example, a table name in DB2 UDB for iSeries can be 128 bytes, but other products may not support table names that are that long. Thus, even if the generated SQL statement is standard, it still may not work on other products if they have smaller limits that those on DB2 Universal Database for iSeries.

If option 1 is specified,

- The naming option must be SQL.
- The date format must be ISO, USA, EUR, or JIS.
- The time format must be ISO, USA, EUR, or JIS.
- The decimal point must be the period.

If option 2 is specified,
- The naming option must be SQL.
- The date format must be ISO.
- The time format must be ISO.
- The decimal point must be the period.
- An ALIAS object type must not be specified.

Statement formatting option. The formatting option used in the generated SQL statements. The valid values are:

0	No additional formatting characters are added to the generated SQL statements.
1	Additional end-of-line characters and tab characters are added to the generated SQL statements.

Time format. The format used for time constants in a generated SQL CREATE TABLE statement. The time format may not apply to date constants that are in ISO, EUR, USA, or JIS format in a CREATE VIEW, CREATE TRIGGER, CREATE FUNCTION, or CREATE PROCEDURE statement. in the generated SQL statements. The valid values are:

ISO	International Standards Organization (hh.mm.ss)
EUR	IBM European Standard (hh.mm.ss) If the Standards option is '2', the EUR time format is not valid.
JIS	Japanese Industrial standard Christian Era (hh:mm:ss) If the Standards option is '2', the JIS time format is not valid.
USA	IBM USA standard (hh:mm AM, hh:mm PM) If the Standards option is '2', the USA time format is not valid.
HMS	Hour/minute/second (hh:mm:ss) If the Standards option is '1' or '2', the HMS time format is not valid.

Time separator. The time separator used for time constants in a generated SQL CREATE TABLE statement. The time separator may not apply to date constants that are in ISO, EUR, USA, or JIS format in a CREATE VIEW, CREATE TRIGGER, CREATE FUNCTION, or CREATE PROCEDURE statement. The time separator is only applicable if the time format is HMS. in the generated SQL statements. The valid values are:

:	Colon separator
.	Period separator
,	Comma separator
blank	Blank separator

Usage Notes

If the value of the statement formatting option is 0, the generated SQL statements will be minimally formatted by adding blanks. For example:

```
CREATE TABLE mjatst.table_one (
  column_one INTEGER,
  column_two INTEGER,
  column_three CHAR(4000));
```

If the value of the statement formatting option is 1, the generated SQL statements will be formatted by inserting end-of-line characters, tab characters, and spaces. For example:

```
CREATE TABLE mjatst.table_one (
        column_one INTEGER,
        column_two INTEGER,
        column_three CHAR(4000));
```

Error Messages

Message ID	**Error Message Text**
CPF24B4 E	Severe error while addressing parameter list.
CPF3C21 E	Format name &1 is not valid.
» CPF3C23 «	Object &1 is not a file of the correct type.
» CPF3C26 «	File &1 has no members.
CPF3C39 E	Value for reserved field not valid.
CPF3C3A E	Value for parameter &2 for API &1 not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPF8100 E	All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9800 E	All CPF98xx messages could be signaled. xx is from 01 to FF.
» SQL0113 «	Name &1 not allowed.
» SQL7001 «	File &1 in &2 not database file.
» SQL7003 «	File &1 in &2 has more than one format.
» SQL7011 «	&1 in &2 not table, view, or physical file.
» SQL7039 «	QSQGNDDL API input template field values not compatible.
» SQL7040 «	Message severity &1 exceeds specified severity level &2.
» SQL7041 «	&1 in &2 not valid for object type &3.
» SQL7042 «	Value not valid for QSQGNDDL API input template.
» SQL7043 «	System-generated function or built-in data type &1 in &2 not allowed.
» SQL7044 «	&3 not supported by the standards option for &1 in &2.
» SQL7045 «	Function &1 in &2 not allowed.
» SQL7046 «	Generate SQL for &1 in &2 object type &3 failed.
» SQL7047 «	System trigger &1 in &2 not allowed.

API introduced: V5R1

Top | "Database and File APIs," on page 1 | APIs by category

Process Extended Dynamic SQL (QSQPRCED) API

Required Parameter Group:

1	SQL communications area
Output	Char(136)
2	SQL descriptor area
Input	Char(*)
3	Function template format
Input	Char(8)
4	Function template
Input	Char(*)
5	Error code
I/O	Char(*)

»

Optional Parameter Group 1:

6	SQL diagnostic information receiver
Output	Char(*)
7	Length of SQL diagnostic information receiver
Input	Binary(4)

«

Default Public Authority: *USE

Threadsafe: Conditional; see “Usage Notes” on page 501.

The Process Extended Dynamic SQL (QSQPRCED) API provides functions to process extended dynamic SQL statements in an SQL package object. In particular, this API provides the user with the only way to do blocked INSERT using SQLDA.

Authorities and Locks

Creating an SQL package requires that you have *ADD and *READ authority to the library that will contain the package. Using an existing SQL package requires that you have *OBJOPR and *READ authority to the package. To use the PREPARE function of the API, you must have *OBJOPR and *ADD authority to the package. To use a sort sequence table, you must have *USE authority to the table and *EXECUTE authority to the library containing the table. To delete a specified package, you must have *OBJEXIST authority to the package and *EXECUTE authority to the library containing the package.

Required Parameter Group

SQL communications area
OUTPUT; CHAR(136)

This is used for returning diagnostic information. It includes the SQLCODE variable, indicating whether an error has occurred. If SQLCODE has a value of 0 after a call to this API, the function was successful.

You should have this space declared in the program that calls this API. This parameter is considered output because the API uses the space to pass back information. The format of the structure is standard and can be included using the INCLUDE SQLCA statement in an SQL program. It is described more completely in the DB2 UDB for iSeries SQL Programming Concepts topic and DB2 UDB for iSeries SQL Reference topic.

SQL descriptor area
INPUT; CHAR(*)

This is used for you to pass information about the variables being used on a specific SQL statement. The SQLDA is used for passing the address, data type, length, and coded character set identifier (CCSID)for variables on an OPEN, EXECUTE, FETCH, or DESCRIBE function.

The format of the structure is standard and can be included using the INCLUDE SQLDA statement in an SQL program. It is described more completely in the DB2 UDB for iSeries SQL Programming Concepts topic and DB2 UDB for iSeries SQL Reference topic.

Function template format
INPUT; CHAR(8)

The format of the function template being used. The possible values are:

SQLP0100	Basic template
» *SQLP0110*	Extended SQLP0100 template «
SQLP0200	Template for using scrollable cursors or blocked INSERT
» *SQLP0210*	Extended SQLP0200 template «
» *SQLP0300*	Template for options that may improve query performance «
» *SQLP0310*	Extended SQLP0300 template «
» *SQLP0400*	Template for specifying additional options that apply to package creation. «
» *SQLP0410*	Extended SQLP0400 template «
» *SQLP0500*	Template for specifying the retrieval of SQL diagnostic information. «

» For more information, see “SQLP0100 Format” on page 478, SQLP0110 Format (page “SQLP0110 Format” on page 479), “SQLP0200 Format” on page 479, SQLP0210 Format (page “SQLP0210 Format” on page 480), “SQLP0300 Format” on page 481, SQLP0310 Format (page “SQLP0310 Format” on page 482), “SQLP0400 Format” on page 484, SQLP0410 Format (page “SQLP0410 Format” on page 485) or SQLP0500 Format (page “SQLP0500 Format” on page 486)

« .

Function template
INPUT; CHAR(*)

A structure that determines the function to perform, the requested statement to process, and the SQL package to be used. This also contains the text of the statement, which is required for the PREPARE function. » For the format of this parameter, see “SQLP0100 Format” on page 478, SQLP0110 Format (page “SQLP0110 Format” on page 479), “SQLP0200 Format” on page 479, SQLP0210 Format (page “SQLP0210 Format” on page 480), “SQLP0300 Format” on page 481, SQLP0310 Format (page “SQLP0310 Format” on page 482), “SQLP0400 Format” on page 484, SQLP0410 Format (page “SQLP0410 Format” on page 485) or SQLP0500 Format (page “SQLP0500 Format” on page 486) « .

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Optional Parameter Group 1

SQL diagnostic information receiver
OUTPUT; CHAR(*)

This is used for receiving SQL diagnostic information from the SQL diagnostic area. The SQL diagnostic area contains information about an SQL statement (other then a GET DIAGNOSTIC statement) that was executed prior to invoking the API.

You should have this space declared in the program that calls this API. This parameter is considered output because the API uses the space to pass back information.

The format of the data received in this space depends on the type of diagnostic item(s) being retrieved. You can specify statement, condition, or connection information item types to be retrieved on SQLP0500 Format (page "SQLP0500 Format" on page 486) template. For detailed description of the format of the retrieved data specific for an information item type, see Diagnostic Information Data Format (page "Diagnostic Information Data Format" on page 496).

This parameter is required for function D.

Length of SQL diagnostic information receiver
INPUT; BINARY(4)

The length of the SQL diagnostic information receiver.

This parameter is required for function D and it must be specified for this function with the minimum value of 8. If specified for other functions, it must be set to 0.

«

SQLP0100 Format

The following shows the format of the function template parameter for the SQLP0100 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 487.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(1)	Function
1	1	CHAR(10)	SQL package name
11	B	CHAR(10)	SQL package library name
21	15	CHAR(10)	Main program name
31	1F	CHAR(10)	Main program library name
41	29	CHAR(18)	Statement name
59	3B	CHAR(18)	Cursor name
77	4D	CHAR(1)	Open options
78	4E	CHAR(1)	Using clause for describe
79	4F	CHAR(1)	Commitment control
80	50	CHAR(3)	Date format
83	53	CHAR(1)	Date separator
84	54	CHAR(3)	Time format
87	57	CHAR(1)	Time separator
88	58	CHAR(3)	Naming option
91	5B	CHAR(1)	Decimal point
92	5C	BINARY(2)	Blocking factor

Offset			
Dec	Hex	Type	Field
94	5E	BINARY(2)	Statement length
96	60	CHAR(*)	Statement text

SQLP0110 Format

The following shows the format of the function template parameter for the SQLP0110 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 487.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(1)	Function
1	1	CHAR(10)	SQL package name
11	B	CHAR(10)	SQL package library name
21	15	CHAR(10)	Main program name
31	1F	CHAR(10)	Main program library name
41	29	CHAR(18)	Statement name
59	3B	CHAR(18)	Cursor name
77	4D	CHAR(1)	Open options
78	4E	CHAR(1)	Using clause for describe
79	4F	CHAR(1)	Commitment control
80	50	CHAR(3)	Date format
83	53	CHAR(1)	Date separator
84	54	CHAR(3)	Time format
87	57	CHAR(1)	Time separator
88	58	CHAR(3)	Naming option
91	5B	CHAR(1)	Decimal point
92	5C	BINARY(2)	Blocking factor
94	5E	BINARY(4)	Offset to statement text length and statement text
98	62	CHAR(1)	Hex literal option
99	63	CHAR(13)	Reserved
*	*	BINARY(2)	Statement length
*	*	CHAR(*)	Statement text

«

SQLP0200 Format

The following shows the format of the function template parameter for the SQLP0200 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 487.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(1)	Function

Offset		Type	Field
Dec	Hex		
1	1	CHAR(10)	SQL package name
11	B	CHAR(10)	SQL package library name
21	15	CHAR(10)	Main program name
31	1F	CHAR(10)	Main program library name
41	29	CHAR(18)	Statement name
59	3B	CHAR(18)	Cursor name
77	4D	CHAR(1)	Open options
78	4E	CHAR(1)	Using clause for describe
79	4F	CHAR(1)	Commitment control
80	50	CHAR(3)	Date format
83	53	CHAR(1)	Date separator
84	54	CHAR(3)	Time format
87	57	CHAR(1)	Time separator
88	58	CHAR(3)	Naming option
91	5B	CHAR(1)	Decimal point
92	5C	BINARY(2)	Blocking factor
94	5E	BINARY(2)	Scrollable option
96	60	BINARY(2)	Position option
98	62	BINARY(4)	Relative record
102	66	BINARY(4)	Number of rows to insert
106	6A	BINARY(2)	Statement length
108	6C	CHAR(*)	Statement text

SQLP0210 Format

The following shows the format of the function template parameter for the SQLP0210 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 487.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(1)	Function
1	1	CHAR(10)	SQL package name
11	B	CHAR(10)	SQL package library name
21	15	CHAR(10)	Main program name
31	1F	CHAR(10)	Main program library name
41	29	CHAR(18)	Statement name
59	3B	CHAR(18)	Cursor name
77	4D	CHAR(1)	Open options
78	4E	CHAR(1)	Using clause for describe
79	4F	CHAR(1)	Commitment control
80	50	CHAR(3)	Date format

Offset		Type	Field
Dec	Hex		
83	53	CHAR(1)	Date separator
84	54	CHAR(3)	Time format
87	57	CHAR(1)	Time separator
88	58	CHAR(3)	Naming option
91	5B	CHAR(1)	Decimal point
92	5C	BINARY(2)	Blocking factor
94	5E	BINARY(2)	Scrollable option
96	60	BINARY(2)	Position option
98	62	BINARY(4)	Relative record
102	66	BINARY(4)	Number of rows to insert
106	6A	BINARY(4)	Offset to statement text length and statement text
110	6E	CHAR(1)	Hex literal option
111	6F	CHAR(17)	Reserved
*	*	BINARY(2)	Statement length
*	*	CHAR(*)	Statement text

«

SQLP0300 Format

The following shows the format of the function template parameter for the SQLP0300 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 487.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(1)	Function
1	1	CHAR(10)	SQL package name
11	B	CHAR(10)	SQL package library name
21	15	CHAR(10)	Main program name
31	1F	CHAR(10)	Main program library name
41	29	CHAR(18)	Statement name
59	3B	CHAR(18)	Cursor name
77	4D	CHAR(1)	Open options
78	4E	CHAR(1)	Using clause for describe
79	4F	CHAR(1)	Commitment control
80	50	CHAR(3)	Date format
83	53	CHAR(1)	Date separator
84	54	CHAR(3)	Time format
87	57	CHAR(1)	Time separator
88	58	CHAR(3)	Naming option
91	5B	CHAR(1)	Decimal point

Offset			
Dec	Hex	Type	Field
92	5C	BINARY(2)	Blocking factor
94	5E	BINARY(2)	Scrollable option
96	60	BINARY(2)	Position option
98	62	BINARY(4)	Relative record
102	66	BINARY(4)	Number of rows to insert
106	6A	CHAR(1)	Direct map
107	6B	CHAR(1)	Reuse SQLDA
108	6C	CHAR(1)	Name check
109	6D	CHAR(1)	Use pointers
110	6E	CHAR(1)	WITH HOLD
111	6F	CHAR(18)	User-defined field
129	81	CHAR(10)	Close file name
139	8B	CHAR(10)	Close library name
149	95	CHAR(1)	Reopen
150	96	CHAR(1)	Use performance area
» 151	97	CHAR(1)	Reserved «
» 152	98	BINARY(2)	Maximum Scale «
» 153	99	CHAR(1)	Maximum Precision «
» 155	9B	CHAR(1)	Minimum Divide Scale «
156	9C	BINARY(4)	Statement text CCSID
160	A0	PTR(SYP)	SQL-package system pointer
176	B0	PTR(SYP)	Main-program system pointer
192	C0	BINARY(2)	Statement length
194	C2	CHAR(*)	Statement text

SQLP0310 Format

The following shows the format of the function template parameter for the SQLP0310 format. For detailed descriptions of the fields in the table, see “Field Descriptions” on page 487.

Offset			
Dec	Hex	Type	Field
1	1	CHAR(10)	SQL package name
11	B	CHAR(10)	SQL package library name
21	15	CHAR(10)	Main program name
31	1F	CHAR(10)	Main program library name
41	29	CHAR(18)	Statement name
59	3B	CHAR(18)	Cursor name
77	4D	CHAR(1)	Open options
78	4E	CHAR(1)	Using clause for describe
79	4F	CHAR(1)	Commitment control

Offset			
Dec	Hex	Type	Field
80	50	CHAR(3)	Date format
83	53	CHAR(1)	Date separator
84	54	CHAR(3)	Time format
87	57	CHAR(1)	Time separator
88	58	CHAR(3)	Naming option
91	5B	CHAR(1)	Decimal point
92	5C	BINARY(2)	Blocking factor
94	5E	BINARY(2)	Scrollable option
96	60	BINARY(2)	Position option
98	62	BINARY(4)	Relative record
102	66	BINARY(4)	Number of rows to insert
106	6A	CHAR(1)	Direct map
107	6B	CHAR(1)	Reuse SQLDA
108	6C	CHAR(1)	Name check
109	6D	CHAR(1)	Use pointers
110	6E	CHAR(1)	WITH HOLD
111	6F	CHAR(18)	User-defined field
129	81	CHAR(10)	Close file name
139	8B	CHAR(10)	Close library name
149	95	CHAR(1)	Reopen
150	96	CHAR(1)	Use performance area
» 151	97	CHAR(1)	Reserved «
» 152	98	BINARY(2)	Maximum Scale «
» 153	99	CHAR(1)	Maximum Precision «
» 155	9B	CHAR(1)	Minimum Divide Scale «
156	9C	BINARY(4)	Statement text CCSID
160	A0	PTR(SYP)	SQL-package system pointer
176	B0	PTR(SYP)	Main-program system pointer
192	C0	BINARY(4)	Offset to statement text length and statement text
196	C4	CHAR(1)	Hex literal option
» 197	C5	CHAR(7)	Reserved «
» 204	CC	BINARY(4)	Length of Additional Fields «
» 208	D0	BINARY(4)	Connection Handle «
» 212	D4	CHAR(1)	Autocommit Option «
*	*	BINARY(2)	Statement length
*	*	CHAR(*)	Statement text

«

SQLP0400 Format

The following shows the format of the function template parameter for the SQLP0400 format. For detailed descriptions of the fields in the table, see “Field Descriptions” on page 487.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(1)	Function
1	1	CHAR(10)	SQL package name
11	B	CHAR(10)	SQL package library name
21	15	CHAR(10)	Main program name
31	1F	CHAR(10)	Main program library name
41	29	CHAR(18)	Statement name
59	3B	CHAR(18)	Cursor name
77	4D	CHAR(1)	Open options
78	4E	CHAR(1)	Using clause for describe
79	4F	CHAR(1)	Commitment control
80	50	CHAR(3)	Date format
83	53	CHAR(1)	Date separator
84	54	CHAR(3)	Time format
87	57	CHAR(1)	Time separator
88	58	CHAR(3)	Naming option
91	5B	CHAR(1)	Decimal point
92	5C	BINARY(2)	Blocking factor
94	5E	BINARY(2)	Scrollable option
96	60	BINARY(2)	Position option
98	62	BINARY(4)	Relative record
102	66	BINARY(4)	Number of rows to insert
106	6A	CHAR(1)	Direct map
107	6B	CHAR(1)	Reuse SQLDA
108	6C	CHAR(1)	Name check
109	6D	CHAR(1)	Use pointers
110	6E	CHAR(1)	WITH HOLD
111	6F	CHAR(18)	User-defined field
129	81	CHAR(10)	Close file name
139	8B	CHAR(10)	Close library name
149	95	CHAR(1)	Reopen
150	96	CHAR(1)	Use performance area
» 151	97	CHAR(1)	Reserved «
» 152	98	BINARY(2)	Maximum Scale «
» 153	99	CHAR(1)	Maximum Precision «
» 155	9B	CHAR(1)	Minimum Divide Scale «
156	9C	BINARY(4)	Statement text CCSID
160	A0	PTR(SYP)	SQL-package system pointer

Offset			
Dec	Hex	Type	Field
176	B0	PTR(SYP)	Main-program system pointer
192	C0	CHAR(10)	Sort sequence table name
202	CA	CHAR(10)	Sort sequence library name
212	D4	CHAR(10)	Language identifier
222	DE	CHAR(1)	Allow copy of data
223	DF	CHAR(1)	Allow blocking
224	E0	BINARY(2)	Statement length
226	E2	CHAR(*)	Statement text

SQLP0410 Format

The following shows the format of the function template parameter for the SQLP0410 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 487.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(1)	Function
1	1	CHAR(10)	SQL package name
11	B	CHAR(10)	SQL package library name
21	15	CHAR(10)	Main program name
31	1F	CHAR(10)	Main program library name
41	29	CHAR(18)	Statement name
59	3B	CHAR(18)	Cursor name
77	4D	CHAR(1)	Open options
78	4E	CHAR(1)	Using clause for describe
79	4F	CHAR(1)	Commitment control
80	50	CHAR(3)	Date format
83	53	CHAR(1)	Date separator
84	54	CHAR(3)	Time format
87	57	CHAR(1)	Time separator
88	58	CHAR(3)	Naming option
91	5B	CHAR(1)	Decimal point
92	5C	BINARY(2)	Blocking factor
94	5E	BINARY(2)	Scrollable option
96	60	BINARY(2)	Position option
98	62	BINARY(4)	Relative record
102	66	BINARY(4)	Number of rows to insert
106	6A	CHAR(1)	Direct map
107	6B	CHAR(1)	Reuse SQLDA
108	6C	CHAR(1)	Name check
109	6D	CHAR(1)	Use pointers

Offset			
Dec	Hex	Type	Field
110	6E	CHAR(1)	WITH HOLD
111	6F	CHAR(18)	User-defined field
129	81	CHAR(10)	Close file name
139	8B	CHAR(10)	Close library name
149	95	CHAR(1)	Reopen
150	96	CHAR(1)	Use performance area
» 151	97	CHAR(1)	Reserved «
» 152	98	BINARY(2)	Maximum Scale «
» 153	99	CHAR(1)	Maximum Precision «
» 155	9B	CHAR(1)	Minimum Divide Scale «
156	9C	BINARY(4)	Statement text CCSID
160	A0	PTR(SYP)	SQL-package system pointer
176	B0	PTR(SYP)	Main-program system pointer
192	C0	CHAR(10)	Sort sequence table name
202	CA	CHAR(10)	Sort sequence library name
212	D4	CHAR(10)	Language identifier
222	DE	CHAR(1)	Allow copy of data
223	DF	CHAR(1)	Allow blocking
224	E0	BINARY(4)	Offset to statement text length and statement text
228	E4	CHAR(1)	Hex literal option
» 229	E5	CHAR(7)	Reserved «
» 236	EC	BINARY(4)	Length of Additional Fields «
» 240	F0	BINARY(4)	Connection Handle «
» 244	F4	CHAR(1)	Autocommit Option «
*	*	BINARY(2)	Statement length
*	*	CHAR(*)	Statement text

«

SQLP0500 Format

The following shows the format of the function template parameter for the SQLP0500 format. For detailed descriptions of the fields in the table, see “Field Descriptions” on page 487.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(1)	Function
1	1	CHAR(3)	Date format
4	4	CHAR(1)	Date separator
5	5	CHAR(3)	Time format
8	8	CHAR(1)	Time separator

Offset			
Dec	Hex	Type	Field
9	9	CHAR(3)	Naming option
12	C	CHAR(1)	Decimal point
13	D	CHAR(3)	Reserved
16	10	BINARY(4)	Diagnostic information CCSID
20	14	BINARY(4)	Condition number
24	18	CHAR(4)	Reserved
28	1C	BINARY(4)	Offset to statement, condition, or connection information item types list
32	20	BINARY(4)	Number of statement, condition, or connection information items
*	*	ARRAY(*) of BINARY(4)	Statement, condition, or connection information item types list

«

Field Descriptions

Allow copy of data. Whether a copy of the data can be used in a SELECT statement. The valid values follow:

A	A copy of the data is used only when necessary.
S	The system determines whether to use the data retrieved directly from the database or to use a copy of the data. The decision is based on which method provides the best performance. If commitment control level is C or S and the Allow Blocking field is not L, or if the commitment control level is A or R, then a copy of the data is used only when it is necessary to run the query.
N	A copy of the data is not allowed. If a temporary copy of the data is required to perform the query, an error message is returned.

The allow copy of data value is required for function 1. It is ignored for other functions.

Allow blocking. Whether the database manager can use record blocking, and the extent to which blocking can be used for read-only cursors. The valid values follow:

S

*READ:
Records are blocked for read-only retrieval of data for cursors the following conditions are met:

- N is specified for the commitment control field, which indicates that commitment control is not used.
- The cursor is declared with a FOR FETCH ONLY clause or there are no dynamic statements that could run a positioned UPDATE or DELETE statement for the cursor.

You can specify S to improve the overall performance of queries that meet the above conditions and retrieve a large number of records.

F

*NONE:
Rows are not blocked for retrieval of data for cursors. If you specify F, the following occurs:

- Guarantees that the data retrieved is current.
- May reduce the amount of time required to retrieve the first row of data for a query.
- Stops the database manager from retrieving a block of data rows that is not used by the program when only the first few rows of a query are retrieved before the query is closed.
- Can degrade the overall performance of a query that retrieves a large number of rows.

L	*ALLREAD: Rows are blocked for read-only cursors if N or C is specified on the commitment control field. All cursors in a program that are not explicitly able to be updated are opened for read-only processing even though EXECUTE or EXECUTE IMMEDIATE statements may be in the program. If you specify L, the following occurs: • Allows record blocking under commitment control level C in addition to the blocking allowed for S. • Can improve the performance of almost all read-only cursors in programs, but limits queries in the following ways: – A ROLLBACK statement or ROLLBACK HOLD SQL statement does not reposition a read-only cursor when L is specified. – Dynamic running of a positioned UPDATE or DELETE statement (for example, using EXECUTE IMMEDIATE) cannot be used to update a row in a cursor unless the DECLARE statement for the cursor includes the FOR UPDATE clause.

The allow blocking value is required for function 1. It is ignored for other functions.

Autocommit Option. When using a connection handle with SQL Server Mode, the autocommit option can be used to direct the database to handle commitment control for the connection. The valid values follow:

'0'	Autocommit is not requested
'1'	Autocommit is requested.

When requested, the database will commit or rollback activity in the QSQSRVR server job after every SQL statement operation sent by the client, when that operation performs some committable work. No commit or rollback occurs after fetch operations. If the SQLCODE indicates a failure, the database will rollback the transaction. If the SQLCODE indicates success, the transaction will be committed.

When Commitment control is set to 'N' (*NONE), the database ignores the autocommit input value and executes the statement without autocommit.

Autocommit will influence functions 3, 4, 5, 6 and 8. It is ignored for other functions.

When the connection handle is set to 0, this field must be set to '0'. This function is only valid for SQLP0310 Format (page "SQLP0310 Format" on page 482) and SQLP0410 Format (page "SQLP0410 Format" on page 485).

Blocking factor. The number of records to be passed on a blocked FETCH request. The same number should be used on the OPEN and the FETCH request. The blocking factor is required for functions 4 and 5. It is ignored for other functions.

Close file name. The name of the file for which all pseudo-closed open data paths should be closed. The file name must be the system file name. It cannot be an SQL long table name. If all pseudo-closed open data paths for the job are to be closed, the close file name and the close library name should be specified as *ALL. The close file name is required for function B. It is ignored for other functions.

If the close library name is *NUMBER or *THRESHOLD, then the first 4 bytes of close file name should contain an integer value. For *NUMBER, the value indicates the number of pseudo-closed cursors to close. For *THRESHOLD, the value indicates the threshold of pseudo-closed cursors that should remain following the closing of pseudo-closed cursors.

Close library name. The library of the close file name. If the close file name is specified as *ALL, the close library name should be *ALL as well. The close library name is required for function B. It is ignored for other functions.

*NUMBER indicates to close a specified number of pseudo-closed cursors. *THRESHOLD indicates to continue closing pseudo-closed cursors until a specified threshold is reached.

Commitment control. The commit level to be used. The possible values are:

C	*CHG
S	*CS
A	*ALL
N	*NONE

The commitment control value is required for function 1. It is ignored for other functions.

» **Condition number.** The number that identifies a condition for which diagnostic information items are to be retrieved from the SQL diagnostic area. See Key Values and Data Types of Condition Diagnostic Information Items (page “Key Values and Data Types of Condition Diagnostic Information Items” on page 499) for a list of the keys of the condition information item types that may be retrieved for the condition. See Key Values and Data Types of Connection Diagnostic Information Items (page “Key Values and Data Types of Connection Diagnostic Information Items” on page 500) for a list of the keys of the connection information item types that may be retrieved for the condition. «

» **Connection Handle.** The connection handle allows SQL Server Mode users to have a one-to-one correspondence with a specific QSQSRVR prestart job in the QSYSWRK subsystem.

When the negated connection handle is passed as input, serialization within the job is not held during the execution of the work handed off to the QSQSRVR job. Using the negated connection handle allows an application to achieve parallel execution across threads which are using different connection handles. If the connection handle were 14, it would be valid to input -14 if the user wanted parallel execution where other threads within this job would be allowed to execute when using a connection handle other than 14.

The Extended Dynamic Remote SQL (EDRS) APIs can be used to establish connections. “Connect to EDRS Server (QxdaConnectEDRS) API” on page 281 (QxdaConnectEDRS) is used to initiate a connection to a server system. The connection handle returned by this API is valid only in the same job and activation group in which it was generated. A connection cannot span multiple jobs or activation groups. The “Extended Dynamic Remote SQL (EDRS) APIs” on page 262 will utilize QSQSRVR jobs when SQL Server Mode is ON and a local connection is requested.

This input field is ignored when SQL Server Mode is not active or when set to 0. SQL Server Mode can be enabled within the job by calling the Change Job (QWTCHGJB) API. When connections are established, completion message SQL7908 is sent to the SQL Server Mode client joblog, indicating which QSQSRVR job is being used for that connection. The commitment definition is owned by the job that is indicated in this message.

This function is only valid for SQLP0310 Format (page “SQLP0310 Format” on page 482) and SQLP0410 Format (page “SQLP0410 Format” on page 485). «

Cursor name. The name of the SQL cursor. The cursor name is required for functions 4, 5, 6, and 8. It is ignored for other functions.

Date format. The format used when accessing date result columns. All output date fields are returned in the format you specify. For input date strings, the value you specify is used to determine whether the date is a valid format. The valid values are:

USA	IBM USA standard (mm.dd.yyyy, hh:mm a.m., hh:mm p.m.)
ISO	International Standards Organization (yyyy-mm-dd, hh.mm.ss)
EUR	IBM European Standard (dd.mm.yyyy, hh.mm.ss)
JIS	Japanese Industrial standard Christian Era (yyyy-mm-dd, hh:mm:ss)
MDY	Month/day/year (mm/dd/yy)
DMY	Day/month/year (dd/mm/yy)
YMD	Year/month/day (yy/mm/dd)
JUL	Julian (yy/ddd)

The date format is required for function 1. » For function D, it must be set to X'000000'. « It is ignored for other functions.

Date separator. The separator used when accessing date result columns. The valid values are:

/	Slash separator
.	Period separator
,	Comma separator
-	Dash separator
blank	Blank separator

The date separator is required for function 1. » For function D, it must be set to X'00'. « It is ignored for other functions.

Decimal point. The decimal point for numeric constants in SQL statements. The valid values are:

.	Period separator
,	Comma separator

The decimal point is required for function 1. » For function D, it must be set to X'00'. « It is ignored for other functions.

» **Diagnostic information CCSID.** CCSID of any CHAR data that is returned for the statement, condition, or connection diagnostic information. If 0 is specified, the data returned will be in the default job CCSID. «

Direct map. Whether the data that is retrieved is to be moved directly into the user area. The possible values follow:

Y	Map the data to the user's area by using a single move operation. SQL obtains the address for the beginning of the user's area from the first SQLDATA entry of the SQLDA. The SQLDA must be set up correctly for all fields in the results list in case the direct map cannot be performed.
N	Use the SQLDA definitions to map the data to the user's area.

The direct map field is optional for function 5. The default value for direct map is N. It is ignored for all other functions.

Function. The function being requested. The possible values follow:

1	Build a new package into the specified library.
2	Prepare a statement into the specified package.

3	Execute a statement from the specified package.
4	Open a cursor defined by a prepared statement in a package.
5	Fetch data from an open cursor.
6	Close an open cursor.
7	Describe a prepared statement in a package.
8	Close an open cursor and delete the open data path
9	Prepare and describe in one step.
A	Inquire as to whether or not a specified statement has been prepared in the specified package.
B	Actually close pseudo-closed cursors.
C	Delete the specified package.
D	Retrieve SQL diagnostic information. This function is only valid for SQLP0500 Format (page “SQLP0500 Format” on page 486).

Hex literal option. Option which allows Hex literals to be treated as binary data instead of treating them as character data. The Hex literal option is used for function 1. It is ignored for all other functions. The valid values follow:

0	Treat Hex literals as character data. The default value is 0.
1	Treat Hex literals as binary data.

Language identifier. The language identifier to be used when *LANGIDUNQ or *LANGIDSHR is specified for the sort sequence table name. The valid values follow:

**JOB*	The language identifier for the job is retrieved when the package is created.
**JOBRUN*	The language identifier for the job is retrieved when the program is run.
language-id	The language identifier to be used by the program.

The language identifier value is required for function 1 when a sort sequence value of *LNGIDUNQ or *LNGIDSHR is specified. It is ignored for other functions.

Length of Additional Fields. When used, this field indicates how much space exists for additional fields in the input template. This field is being used to allow the template to be extended to include additional input fields. Set this length to indicate the additional space being passed.

For example, to pass the Connection Handle and Autocommit Option input fields, the length should be set to at least 5. When the length exceeds the size of any defined fields, the additional input storage must be set to hex zeros.

This function is only valid for SQLP0310 Format (page “SQLP0310 Format” on page 482) and SQLP0410 Format (page “SQLP0410 Format” on page 485).

Main program library name. The library of the main program.

Main program name. The name of the program or service program representing the top program in the SQL application. When this program completes, all cursors are closed and the SQL environment goes away. This program must be on the stack or an error will occur (SQL0901). The main program name is required for all functions except 1. This allows you to control the boundary of the application. If you want to scope to an activation group, as opposed to the main program name, this can be done by specifying *ENDACTGRP for the main program name. This special value is only allowed for function 1. For all other functions, specify the actual main program name.

Main-program system pointer. A system pointer that has been resolved to point to the main program. This field is ignored if the use pointers field has not been set to Y. If the use pointers field is specified, this field is used in place of the main program name and main program library name.

» **Maximum Precision.** Specifies the maximum precision (length) that should be used for decimal operations. The possible values follow:

1	Maximum precision is 31.
2	Maximum precision is 63.

The Maximum Precision is optional for function 1. The default is 1. It is ignored for all other functions.

Maximum Scale. Specifies the maximum scale (number of decimal positions to the right of the decimal point) that should be used for decimal operations. The value can range from 0 to the Maximum Precision.

The Maximum Scale is optional for function 1. The default is 31. It is ignored for all other functions.

Minimum Divide Scale. Specifies the minimum divide scale (number of decimal positions to the right of the decimal point) that should be used for both intermediary and result data types. The value can range from ’0’ to ’9’ and may not exceed the Maximum Scale.

The Minimum Divide Scale is optional for function 1. The default is ’0’. It is ignored for all other functions.

«

Name check. Whether the statement names and cursor names are to be completely checked for valid name syntax. The possible values follow:

Y	Check the names for valid name syntax.
N	Do not check the names for valid syntax.

The name check field is optional. The default value for name check is Y. It is ignored for functions 1 and B.

Naming option. The naming convention used for naming objects in SQL statements. The valid values are:

SYS	*library/file* syntax
SQL	*collection/table* syntax

The naming option is required for function 1. » For function D, it must be set to X’000000’. « It is ignored for other functions.

Number of rows to insert. When you request an INSERT statement, this value indicates how many rows are being inserted. Blocked INSERT using SQLDA is similar to blocked FETCH using SQLDA. Refer to the DB2 UDB for iSeries SQL Reference topic for instructions on how to set up the SQLDA to do blocked FETCH. Refer to “Blocked INSERT Using SQLDA Setup Requirements” on page 501 for blocked INSERT requirements that are different from blocked FETCH.

The prepared INSERT statement must be a blocked INSERT with a parameter marker specified for the number of rows.

The number of rows to insert is required for function 3 but is used only when the statement is an INSERT. It is ignored for all other functions.

Number of statement, condition, or connection information items. The number of items specified in the statement, condition, or connection information item types list. If 0 is specified, no information items will be returned.

Offset to statement, condition, or connection information item type list. Offset from beginning of SQLP0500 Format (page "SQLP0500 Format" on page 486) to the list of statement, condition, or connection information item types.

Offset to statement text length and statement text. Offset from beginning of SQLP00110 Format (page "SQLP0110 Format" on page 479), SQLP00210 Format (page "SQLP0210 Format" on page 480), SQLP00310 Format (page "SQLP0310 Format" on page 482), or SQLP00410 Format (page "SQLP0410 Format" on page 485) to the start of the statement text length field. The statement text should immediately follow the statement text length. This must be 0 for all functions other than 2, 3, 4, 7, 9, and A.

Open options. The open options used on an SQL cursor. These are specified using the following bits:

Bit(0)	Read
Bit(1)	Write
Bit(2)	Update
Bit(3)	Delete

For example, if a cursor is only for FETCH statements, the bit pattern should be '10000000'B or hex 80. If update capability is needed, the bit pattern should be '10100000'B. The syntax in the SQL statement takes precedence over the open options. This means that the FOR UPDATE OF and FOR FETCH ONLY clauses will be honored, even if they do not coincide with the requested open options. The open options are required for functions 2 and 4. They are ignored for other functions.

Position option. The positioning option that is used for a FETCH statement. For options other than NEXT, the cursor must have been opened as a scrollable cursor. The valid options are:

0	FETCH NEXT
1	FETCH PRIOR
2	FETCH FIRST
3	FETCH LAST
5	FETCH BEFORE
6	FETCH AFTER
6	FETCH CURRENT
7	FETCH RELATIVE

The position option is required for function 5. It is ignored for other functions.

Relative record. The number of rows forward or backward to move before retrieving data. A positive number means forward and a negative number, backward. This is required when using function 5 (FETCH) with a position option of FETCH RELATIVE. It is ignored for other options.

Reopen. Whether to allow a cursor that is currently open to be reopened. A reopen operation implicitly closes and opens the cursor. If a reopen operation is requested on a cursor that is currently closed, only an open operation is performed (no implicit close takes place). The valid values follow:

0	Do not allow an open cursor to be reopened.
1	Allow an open cursor to be reopened.

The reopen field is optional for function 4 with a default of 0. It is ignored for all other functions.

Use performance area. Use a performance area internally to store information about the invocation environment. This option is beneficial in environments where statements are run repeatedly. The valid values follow:

0	Do not use the internal performance area. The default value is 0.
1	Use the internal performance area.

Reserved. » All reserved fields must be set to X'00'. «

Reuse SQLDA. Whether the SQLDA is being used again without changes. The possible values follow:

Y	SQLDA is being reused without changes. Do not validate the SQLDA.
N	SQLDA is not being reused. Validate the SQLDA.

The reuse SQLDA field is optional for functions 3, 4, and 5. The default value for reuse SQLDA is N. It is ignored for all other functions.

Scrollable option. Specified if the cursor is scrollable. The cursor must be opened as scrollable if any FETCH options other than FETCH NEXT are used. The valid values are:

0	Cursor is not scrollable
1	Cursor is scrollable

The scrollable option is required for function 4. It is ignored for other functions.

Sort sequence table name. The sort sequence table name to be used for string comparisons in SQL statements. The possible values follow:

**JOB*	The sort sequence value for the job is retrieved when the package is created.
**JOBRUN*	The sort sequence value for the job is retrieved when the program is run.
**LANGIDUNQ*	The unique-weight sort table for the language that is specified on the language identifier field is used.
**LANGIDSHR*	The shared-weight sort table for the language that is specified on the language identifier field is used.
**HEX*	A sort sequence table is not used. The hexadecimal values of the characters are used to determine the sort sequence.
table-name	The name of the sort sequence table to be used.

The sort sequence table name value is required for function 1. It is ignored for other functions.

Sort sequence library name. The name of the sort sequence table can be qualified by one of the following library values:

**LIBL*	All libraries in the job's library list are searched until the first match is found.
**CURLIB*	The current library for the job is searched. If no library is specified as the current library for the job, the QGPL library is used.
library-name	The name of the library to be searched.

The sort sequence library name value is required for function 1 when a table name is specified for the sort sequence table name value. It is ignored for other functions.

SQL package library name. The library of the package.

SQL package name. The name of the SQL package used as the repository for the extended dynamic SQL statements. The SQL package must not be a distributed SQL package created through the Create SQL Package (CRTSQLPKG) or the Create SQL xxx (CRTSQLxxx) commands. Attempted use of a distributed SQL package results in SQL0827. The SQL package name is required for all functions. Function 1 checks the specified package name for valid name syntax. An invalid name results in SQL7023.

SQL-package system pointer. A system pointer that has been resolved to point to the SQL package. This option is ignored if the use pointers field has not been set to Y. If the use pointers field is specified, this field is used in place of the SQL package name and SQL package library name.

» **Statement, condition, or connection information item types list.** The list of statement, condition, or connection information types that are to be retrieved from the SQL diagnostic area. A unique key identifies each item type. For the list of the keys that may be specified for item types, see Key Values and Data Types of Statement Diagnostic Information Items (page “Key Values and Data Types of Statement Diagnostic Information Items” on page 498), Key Values and Data Types of Condition Diagnostic Information Items (page “Key Values and Data Types of Condition Diagnostic Information Items” on page 499), or Key Values and Data Types of Connection Diagnostic Information Items (page “Key Values and Data Types of Connection Diagnostic Information Items” on page 500).

Any one or all of the individual diagnostic information item types may be specified in the list. A default value will be returned if the diagnostic item is currently not set in the SQL diagnostic area. «

Statement length. The length of the SQL statement text that follows. The statement length is required for function 2, 9 and A. It is ignored for other functions.

Statement name. The name of the prepared SQL statement. The statement name is required for functions 2, 3, 4, 7, 9, and A. It is ignored for other functions.

Statement text. The SQL statement text that will be prepared. The statement text is required for function 2. It is ignored for other functions.

» **Statement text CCSID.** The CCSID of the SQL statement text that will be prepared in this package. The statement text CCSID is optional for function 1. It is ignored for other functions. If the SQLP0100, SQLP0110, SQLP0200 or SQLP0210 formats are specified or if statement text CCSID is 0, the job CCSID is used. «

Time format. The format used when accessing time result columns. All output time fields are returned in the format you specify. For input time strings, the value you specify is used to determine whether the time is a valid format. The valid values are:

HMS	Hour/minute/second (hh:mm:ss)
USA	IBM USA standard (mm.dd.yyyy, hh:mm a.m., hh:mm p.m.)
ISO	International Standards Organization (yyyy-mm-dd, hh.mm.ss)
EUR	IBM European Standard (dd.mm.yyyy, hh.mm.ss)
JIS	Japanese Industrial standard Christian Era (yyyy-mm-dd, hh:mm:ss)

The time format is required for function 1. » For function D, it must be set to X’000000’. « It is ignored for other functions.

Time separator. The separator used when accessing time result columns. The valid values are:

:	Colon separator
.	Period separator
,	Comma separator
blank	Blank separator

The time separator is required for function 1. » For function D, it must be set to X'00'. « It is ignored for other functions.

Use pointers. Whether the system pointers should be used to locate the main program and the SQL package instead of the symbolic names. The possible values follow:

0	Do not use pointers to the main program and the SQL package. The symbolic names are used to resolve to the objects.
1	Use the main-program and SQL-package system pointers instead of symbolic names. If 1 is specified, the pointers must address the main program and SQL package. The symbolic names are ignored. If 1 is specified, both pointers must be set.

The use pointers field is optional for all functions. The default value for the use pointers field is 0.

User-defined field. Up to 18 bytes of user-defined data that is inserted into the database performance monitor table. The data is only written to the table if you are collecting database performance monitor statistics by using the Start Database Monitor (STRDBMON) or the Start Performance Monitor (STRPFRMON) command. The user-defined field is optional for all functions. If this field is desired when you collect data, you should use it consistently for all functions.

Using clause for describe. The value to assign to each SQLNAME variable in the SQLDA. The possible values are:

N	Column names
L	Column labels
B	Both (SQLDA must be allocated for twice as many entries)
A	Any labels that exist

These are explained more completely in the DB2 UDB for iSeries SQL Reference topic. The using clause is required for functions 7 and 9. It is ignored for other functions.

DLYPRP (delay PREPARE) is an option on an SQL precompile operation that cannot be specified on the creation of a package (function 1). DLYPRP(*NO) is used as the default.

Refer to the DB2 Universal Database for iSeries documentation for a full description of all the options.

WITH HOLD. Whether the WITH HOLD SQL option should be applied to the statement. The possible values follow:

Y	The cursor is not closed as a consequence of a commit operation. The commit operation commits all the changes in the current unit of work but releases only locks that are not required to maintain the cursor.
N	The cursor is closed at the time of commit.

The WITH HOLD field is optional for functions 2 and 9. The default for WITH HOLD is N. It is ignored for all other functions.

Diagnostic Information Data Format

The following shows the format of the data returned in the SQL diagnostic information receiver when a statement, condition, or connection information item data is requested. For detailed descriptions of the fields in the table, see Diagnostic Information Field Descriptions (page "Diagnostic Information Field Descriptions" on page 497).

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Number of diagnostic information items returned
12	C	ARRAY(*) of CHAR(*)	Diagnostic information items

The following shows the format of the diagnostic information item data returned in the SQL diagnostic information receiver for each diagnostic information item. For detailed descriptions of the fields in the table, see Diagnostic Information Field Descriptions (page “Diagnostic Information Field Descriptions”).

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Length of diagnostic information item
4	4	BINARY(4)	Key of diagnostic information item
8	8	Char(1)	Type of data returned
9	9	Char(3)	Reserved
12	C	BINARY(4)	Length of data returned
16	10	Char(*)	Data returned

Diagnostic Information Field Descriptions

Bytes available. The number of bytes of data available to be returned. All available data is returned if enough space is provided.

Bytes returned. The number of bytes of data returned.

Diagnostic information item. Diagnostic information item data returned in the SQL diagnostic information receiver for each diagnostic information item. For detailed description of the format of the returned data, see Diagnostic Information Data Format (page “Diagnostic Information Data Format” on page 496).

Data returned. Data returned for the diagnostic item identified by the key. Refer to the DB2 UDB for iSeries SQL Reference for information on the possible data value for the diagnostic item.

Key of diagnostic information item. Key that identifies the returned diagnostic information item. For a list of key values for each diagnostic information item available for retrieval, see Key Values and Data Types of Statement Diagnostic Information Items (page “Key Values and Data Types of Statement Diagnostic Information Items” on page 498), Key Values and Data Types of Condition Diagnostic Information Items (page “Key Values and Data Types of Condition Diagnostic Information Items” on page 499), or Key Values and Data Types of Connection Diagnostic Information Items (page “Key Values and Data Types of Connection Diagnostic Information Items” on page 500).

Length of data returned. Length of data returned for the item.

Length of diagnostic information item. Length of the information returned for the item. This length can be used to access the next diagnostic information item.

Number of diagnostic information items returned. Number of diagnostic information returned in the space provided.

Reserved. A field that can be ignored.

Type of data returned. Type of the data returned for the item. The possible values are:

B	Binary data
C	Character data
D	Decimal data

Key Values and Data Types of Statement Diagnostic Information Items

Key	Type	SQL Get Diagnostics Item
101	CHAR(*)	COMMAND_FUNCTION
102	BINARY(4)	COMMAND_FUNCTION_CODE
103	BINARY(4)	DB2_RELATIVE_COST_ESTIMATE
104	BINARY(4)	DB2_DIAGNOSTIC_CONVERSION_ERROR
105	CHAR(*)	DB2_GET_DIAGNOSTICS_DIAGNOSTICS
106	BINARY(4)	DB2_LAST_ROW
107	BINARY(4)	DB2_NUMBER_CONNECTIONS
108	BINARY(4)	DB2_NUMBER_PARAMETER_MARKERS
109	BINARY(4)	DB2_NUMBER_RESULT_SETS
110	DECIMAL(31,0)	DB2_NUMBER_ROWS
111	BINARY(4)	DB2_NUMBER_SUCCESSFUL_SUBSTMTS
112	BINARY(4)	DB2_RETURN_STATUS
113	DECIMAL(31,0)	DB2_ROW_COUNT_SECONDARY
114	BINARY(4)	DB2_ROW_LENGTH
115	CHAR(1)	DB2_SQL_ATTR_CONCURRENCY
116	CHAR(1)	DB2_SQL_ATTR_CURSOR_CAPABILITY
117	CHAR(1)	DB2_SQL_ATTR_CURSOR_HOLD
118	CHAR(1)	DB2_SQL_ATTR_CURSOR_ROWSET
119	CHAR(1)	DB2_SQL_ATTR_CURSOR_SCROLLABLE
120	CHAR(1)	DB2_SQL_ATTR_CURSOR_SENSITIVITY
121	CHAR(1)	DB2_SQL_ATTR_CURSOR_TYPE
122	CHAR(*)	DYNAMIC_FUNCTION
123	BINARY(4)	DYNAMIC_FUNCTION_CODE
124	CHAR(1)	MORE
125	BINARY(4)	NUMBER
126	DECIMAL(31,0)	ROW_COUNT
127	BINARY(4)	TRANSACTION_ACTIVE
128	BINARY(4)	TRANSACTIONS_COMMITTED
129	BINARY(4)	TRANSACTIONS_ROLLED_BACK

Key Values and Data Types of Condition Diagnostic Information Items

Key	Type	SQL Get Diagnostics Item
201	CHAR(*)	CATALOG_NAME
202	CHAR(*)	CLASS_ORIGIN
203	CHAR(*)	COLUMN_NAME
204	CHAR(*)	CONDITION_IDENTIFIER
205	BINARY(4)	CONDITION_NUMBER
206	CHAR(*)	CONSTRAINT_CATALOG
207	CHAR(*)	CONSTRAINT_NAME
208	CHAR(*)	CONSTRAINT_SCHEMA
209	CHAR(*)	CURSOR_NAME
210	BINARY(4)	DB2_ERROR_CODE1
211	BINARY(4)	DB2_ERROR_CODE2
212	BINARY(4)	DB2_ERROR_CODE3
213	BINARY(4)	DB2_ERROR_CODE4
214	BINARY(4)	DB2_INTERNAL_ERROR_POINTER
215	BINARY(4)	DB2_LINE_NUMBER
216	CHAR(10)	DB2_MESSAGE_ID
217	CHAR(*)	DB2_MESSAGE_ID1
218	CHAR(*)	DB2_MESSAGE_ID2
219	BINARY(4)	DB2_MESSAGE_KEY
220	CHAR(*)	DB2_MODULE_DETECTING_ERROR
221	BINARY(4)	DB2_NUMBER_FAILING_STATEMENTS
222	BINARY(4)	DB2_OFFSET
223	CHAR(*)	DB2_ORDINAL_TOKEN_1
224	CHAR(*)	DB2_ORDINAL_TOKEN_2
225	CHAR(*)	DB2_ORDINAL_TOKEN_3
226	CHAR(*)	DB2_ORDINAL_TOKEN_4
227	CHAR(*)	DB2_ORDINAL_TOKEN_5
228	CHAR(*)	DB2_ORDINAL_TOKEN_6
229	CHAR(*)	DB2_ORDINAL_TOKEN_7
230	CHAR(*)	DB2_ORDINAL_TOKEN_8
231	CHAR(*)	DB2_ORDINAL_TOKEN_9
232	CHAR(*)	DB2_ORDINAL_TOKEN_10
233	CHAR(*)	DB2_ORDINAL_TOKEN_11
234	CHAR(*)	DB2_ORDINAL_TOKEN_12
235	CHAR(*)	DB2_ORDINAL_TOKEN_13
236	BINARY(4)	DB2_PARTITION_NUMBER
237	BINARY(4)	DB2_REASON_CODE
238	BINARY(4)	DB2_RETURNED_SQLCODE
239	BINARY(4)	DB2_ROW_NUMBER

Key	Type	SQL Get Diagnostics Item
240	CHAR(1)	DB2_SQLERRD_SET
241	BINARY(4)	DB2_SQLERRD1
242	BINARY(4)	DB2_SQLERRD2
243	BINARY(4)	DB2_SQLERRD3
244	BINARY(4)	DB2_SQLERRD4
245	BINARY(4)	DB2_SQLERRD5
246	BINARY(4)	DB2_SQLERRD6
247	BINARY(4)	DB2_TOKEN_COUNT
248	CHAR(70)	DB2_TOKEN_STRING
249	BINARY(4)	MESSAGE_LENGTH
250	BINARY(4)	MESSAGE_OCTET_LENGTH
251	CHAR(*)	MESSAGE_TEXT
252	CHAR(*)	PARAMETER_MODE
253	CHAR(*)	PARAMETER_NAME
254	BINARY(4)	PARAMETER_ORDINAL_POSITION
255	CHAR(5)	RETURNED_SQLSTATE
256	CHAR(*)	ROUTINE_CATALOG
257	CHAR(*)	ROUTINE_NAME
258	CHAR(*)	ROUTINE_SCHEMA
259	CHAR(*)	SCHEMA_NAME
260	CHAR(*)	SERVER_NAME
261	CHAR(*)	SPECIFIC_NAME
262	CHAR(*)	SUBCLASS_ORIGIN
263	CHAR(*)	TABLE_NAME
264	CHAR(*)	TRIGGER_CATALOG
265	CHAR(*)	TRIGGER_NAME
266	CHAR(*)	TRIGGER_SCHEMA «

Key Values and Data Types of Connection Diagnostic Information Items

Key	Type	SQL Get Diagnostics Item
301	CHAR(*)	GET DIAGNOSTICS CONNECTION_NAME
302	CHAR(1)	DB2_AUTHENTICATION_TYPE
303	CHAR(*)	DB2_AUTHORIZATION_ID
304	CHAR(1)	DB2_CONNECTION_METHOD
305	BINARY(4)	DB2_CONNECTION_NUMBER
306	BINARY(4)	DB2_CONNECTION_STATE
307	BINARY(4)	DB2_CONNECTION_STATUS
308	BINARY(2)	DB2_CONNECTION_TYPE
309	BINARY(4)	DB2_DYN_QUERY_MGMT

Key	Type	SQL Get Diagnostics Item
310	CHAR(1)	DB2_ENCRYPTION_TYPE
311	CHAR(8)	DB2_PRODUCT_ID
312	CHAR(*)	DB2_SERVER_CLASS_NAME
313	CHAR(*)	DB2_SERVER_NAME

«

Blocked INSERT Using SQLDA Setup Requirements

Just as in the case of blocked FETCH, the support for blocked INSERT with SQLDA expects the users to have two contiguous areas. One is for the data and the other is for the indicators. The former contains rows of data (the number of rows is given on function 3 calls), and the latter contains rows of indicators.

If none of the columns is null capable, there is no need to have an indicator area. If any of the columns is null capable, all the columns should be turned into null capable (that is, sqltype in all the sqlvar entries should be an odd number), and the row indicator area should have as many indicators per row as there are columns.

In the SQLDA, the pointer sqldata in all the sqlvar entries should be pointing at the data elements for the first row. Similarly, the pointer sqlind in all the sqlvar entries should be pointing at the indicators for the first row, except in the case where there are no null-capable columns at all.

Usage Notes

This function is not threadsafe when called in the following way:

- Using a Data Definition Language (DDL) SQL statement, for example: CREATE, DROP or ALTER.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C21 E	Format name &1 is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
SQL0204 E	&1. and &2 type &3 not found.
SQL0516 E	Prepare statement &2 not found.
SQL0901 E	SQL system error.
SQL7023 E	Parameter value not valid.

API introduced: V2R3

Top | "Database and File APIs," on page 1 | APIs by category

sqludf_append()—SQL LOB Append to Locator

Syntax

```
#include <sqludf.h>

extern int SQL_API_FN sqludf_append(
   udf_locator * udfloc_p,
   unsigned char * buffer_p,
   long          length,
   long *        return_len_p)
```

Service Program Name: QSYS/QSQAPIS

Default Public Authority: *USE

Threadsafe: Yes

The **sqludf_append()** function appends data to the end of the LOB data the locator represents.

Parameters

udfloc_p
(Input) Pointer to the LOB locator value.

buffer_p
(Input) Pointer to the buffer containing the data to append to the end of the LOB data represented by the locator.

length (Input) The number of bytes to append.

return_len_p
(Input/Output) Pointer to the number of bytes actually appended.

Authorities

No authorization is required.

Return Value

sqludf_append()

returns an integer. Possible values are:

0	**sqludf_append()** was successful. The information is returned in the buffer pointed to by *return_len_p*.
-3	**sqludf_append()** was not successful. An invalid parameter was passed into the function.
-137	**sqludf_append()** was not successful. The resulting length of the append exceeds the maximum allowed. The maximum length is 2147483647 bytes.
-423	**sqludf_append()** was not successful. The *udfloc_p* parameter points to an invalid locator value.
-901	**sqludf_append()** was not successful. An SQL system error has occurred.
-7034	**sqludf_append()** was not successful. LOB locators are not allowed with COMMIT(*NONE).

Error Messages

Message ID	Error Message Text
SQL7034 D	LOB locators are not allowed with COMMIT(*NONE).
SQL0901 D	SQL system error.
SQL0952 D	Processing of the SQL statement ended.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Usage Notes

1. This API is used to append data to the end of the LOB data represented by the locator.

Related Information

- "sqludf_create_locator()—SQL LOB Create Locator" on page 505—SQL LOB create locator
- "sqludf_create_locator_with_ccsid()—SQL LOB Create Locator With CCSID" on page 509—SQL LOB create locator with ccsid
- "sqludf_free_locator()—SQL LOB Free Locator" on page 513—SQL LOB free locator
- "sqludf_length()—SQL LOB locator length" on page 515—SQL LOB locator length
- "sqludf_substr()—SQL LOB Substring Locator" on page 519—SQL LOB substring locator

Example

See Code disclaimer information for information pertaining to code examples.

This UDF takes a locator for an input LOB, and returns a locator for another LOB which is a subset of the input LOB. There are some criteria passed as a second input value, which tell the UDF how exactly to break up the input LOB.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sql.h>
#include <sqludf.h>

void SQL_API_FN lob_subsetter(
          udf_locator * lob_input,   /* locator of LOB value to carve up */
          char        * criteria,    /* criteria for carving */
          udf_locator * lob_output,  /* locator of result LOB value */
          sqlint16 * inp_nul,
          sqlint16 * cri_nul,
          sqlint16 * out_nul,
          char     * sqlstate,
          char     * funcname,
          char     * specname,
          char     * msgtext ) {

       /* local vars */
       short j;              /* local indexing var */
       int   rc;             /* return code variable for API calls */
       sqlint32  input_len;  /* receiver for input LOB length */
       sqlint32  input_pos;  /* current position for scanning input LOB */
       char lob_buf[100];    /* data buffer */
       sqlint32  input_rec;  /* number of bytes read by sqludf_substr */
       sqlint32  output_rec; /* number of bytes written by sqludf_append */

       /*---------------------------------------------
        * UDF Program Logic Starts Here
        *---------------------------------------------
        * What we do is create an output handle, and then
        * loop over the input, 50 bytes at a time.
        * Depending on the "criteria" passed in, we may decide
        * to append the 50 byte input lob segment to the output, or not.
        *--------------------------------------------
        * Create the output locator, right in the return buffer.
        */

       rc = sqludf_create_locator(SQL_TYP_CLOB, &lob_output);
       /* Error and exit if unable to create locator */
       if (rc) {
          memcpy (sqlstate, "38901", 5);
          /* special sqlstate for this condition */
          goto exit;
       }
       /* Find out the size of the input LOB value */
       rc = sqludf_length(lob_input, &input_len) ;
       /* Error and exit if unable to find out length */
       if (rc) {
          memcpy (sqlstate, "38902", 5);
          /* special sqlstate for this condition */
          goto exit;
       }
       /* Loop to read next 50 bytes, and append to result if it meets
        * the criteria.
        */
       for (input_pos = 1; (input_pos < input_len); input_pos += 50) {
         /* Read the next 50 (or less) bytes of the input LOB value */
         rc = sqludf_substr(lob_input, input_pos, 50,
                           (unsigned char *) lob_buf, &input_rec) ;
         /* Error and exit if unable to read the segment */
         if (rc) {
            memcpy (sqlstate, "38903", 5);
            /* special sqlstate for this condition */
            goto exit;
         }
         /* apply the criteria for appending this segment to result
   * if (...predicate involving buffer and criteria...) {
   * The example shows if the segment matches the first 6
          * characters with the criteria it is appended.
   */
         if (memcmp(lob_buf,criteria,6) == 0) {
            rc = sqludf_append(lob_output,
                       (unsigned char *) lob_buf, input_rec, &output_rec) ;
            /* Error and exit if unable to read the 50 byte segment */
            if (rc) {
               memcpy (sqlstate, "38904", 5);
               /* special sqlstate for this condition */
               goto exit;
    }
  }
         /* } end if criteria for inclusion met */
       } /* end of for loop, processing 50-byte chunks of input LOB
          * if we fall out of for loop, we are successful, and done.
   */
       *out_nul = 0;
       exit: /* used for errors, which will override null-ness of output. */
       return;
       }
```

Referring to this UDF code, observe that:

- There are includes for *sql.h*, where the type `SQL_TYP_CLOB` used in the `sqludf_create_locator()` call is defined, and `sqludf.h`, where the type `udf_locator` is defined.
- The first input argument, and the third input argument (which represents the function output) are defined as pointers to `sqludf_locator`, that is, they represent CREATE FUNCTION specifications of AS LOCATOR.
- The UDF does not test whether either input argument is null, as NOT NULL CALL is specified in the CREATE FUNCTION statement.

- In the event of error, the UDF exits with `sqlstate` set to 38*xxx*. This is sufficient to stop the execution of the statement referencing the UDF. The actual 38*xxx* SQLSTATE values you choose are not important to DB2, but can serve to differentiate the exception conditions which your UDF may encounter.
- By using the `input_rec` variable as the length of the data appended, the UDF takes care of any partial buffer condition.

Following is the CREATE FUNCTION statement for this UDF:

```
CREATE FUNCTION carve(CLOB(50M) AS LOCATOR, VARCHAR(255) )
  RETURNS CLOB(50M) AS LOCATOR
  NOT NULL CALL
  DETERMINISTIC
  NO SQL
  NO EXTERNAL ACTION
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  EXTERNAL NAME 'MYLIB/LOBUDFS(lob_subsetter)' ;
```

Referring to this statement, observe that:

- NOT NULL CALL is specified, so the UDF will not be called if any of its input SQL arguments are NULL, and does not have to check for this condition.
- The function is specified as DETERMINISTIC, meaning that with a given input CLOB value and a given set of criteria, the result will be the same every time.

Now you can successfully run the following statement:

```
strcpy(hvchar,"return this text 1                                 "
              "remove 1                                           "
              "return this text 2                                 "
              "remove 2                                           ");
exec sql set :hvloc = clob(:hvchar);
exec sql set :hvloc2 = carve(:hvloc,'return');
strcpy(hvchar,"");
exec sql set :hvchar = char(:hvloc2);
```

The UDF is used to subset the value represented by the host variable :hvchar. The first and third 50 byte character segments are returned from the UDF.

API introduced: V5R3

Top | Database and File APIs | APIs by category

sqludf_create_locator()—SQL LOB Create Locator

Syntax

```
#include <sql.h>
#include <sqludf.h>

extern int SQL_API_FN sqludf_create_locator(
   int           loc_type,
   udf_locator ** loc_p)
```

Service Program Name: QSYS/QSQAPIS

Default Public Authority: *USE

Threadsafe: Yes

The **sqludf_create_locator()** function creates a LOB locator.

Parameters

loc_type
(Input) Type of LOB the locator represents. Valid locator types can be any of the types from sql.h representing LOBs. For example:

Type Name	Type Value	Description
SQL_TYP_BLOB	404	BLOB locator
SQL_TYP_NBLOB	405	BLOB locator that allows a null value
SQL_TYP_CLOB	408	CLOB locator
SQL_TYP_NCLOB	409	CLOB locator that allows a null value
SQL_TYP_DBCLOB	412	DBCLOB locator
SQL_TYP_NDBCLOB	413	DBCLOB locator that allows a null value

loc_p (Input/Output) Pointer to a pointer where the locator value is to be returned.

Authorities

No authorization is required.

Return Value

sqludf_create_locator()

returns an integer. Possible values are:

0 **sqludf_create_locator()** was successful. The information is returned in the buffer pointed to by *return_len_p*.

-3 **sqludf_create_locator()** was not successful. An invalid parameter was passed into the function.

-429 **sqludf_create_locator()** was not successful. The maximum number of concurrent LOB locators has been reached.

-901 **sqludf_create_locator()** was not successful. An SQL system error has occurred.

-7034 **sqludf_create_locator()** was not successful. LOB locators are not allowed with COMMIT(*NONE).

Error Messages

Message ID	Error Message Text
SQL7034 D	LOB locators are not allowed with COMMIT(*NONE).
SQL0901 D	SQL system error.
SQL0952 D	Processing of the SQL statement ended.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Usage Notes

1. This API is used to create a locator. A LOB locator is only a mechanism used to refer to a LOB value during a transaction; it does not persist beyond the transaction in which it was created unless it is used with the HOLD LOCATOR statement.

Related Information

- "sqludf_append()—SQL LOB Append to Locator" on page 502—SQL LOB append locator

- “sqludf_create_locator_with_ccsid()—SQL LOB Create Locator With CCSID” on page 509—SQL LOB create locator with ccsid
- “sqludf_free_locator()—SQL LOB Free Locator” on page 513—SQL LOB free locator
- “sqludf_length()—SQL LOB locator length” on page 515—SQL LOB locator length
- “sqludf_substr()—SQL LOB Substring Locator” on page 519—SQL LOB substring locator

Example

See Code disclaimer information for information pertaining to code examples.

This UDF takes a locator for an input LOB, and returns a locator for another LOB which is a subset of the input LOB. There are some criteria passed as a second input value, which tell the UDF how exactly to break up the input LOB.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sql.h>
#include <sqludf.h>

void SQL_API_FN lob_subsetter(
          udf_locator * lob_input,   /* locator of LOB value to carve up */
          char        * criteria,    /* criteria for carving */
          udf_locator * lob_output,  /* locator of result LOB value */
          sqlint16 * inp_nul,
          sqlint16 * cri_nul,
          sqlint16 * out_nul,
          char     * sqlstate,
          char     * funcname,
          char     * specname,
          char     * msgtext ) {

       /* local vars */
       short j;              /* local indexing var */
       int   rc;             /* return code variable for API calls */
       sqlint32  input_len;  /* receiver for input LOB length */
       sqlint32  input_pos;  /* current position for scanning input LOB */
       char lob_buf[100];    /* data buffer */
       sqlint32  input_rec;  /* number of bytes read by sqludf_substr */
       sqlint32  output_rec; /* number of bytes written by sqludf_append */

       /*---------------------------------------------
        * UDF Program Logic Starts Here
        *---------------------------------------------
        * What we do is create an output handle, and then
        * loop over the input, 50 bytes at a time.
        * Depending on the "criteria" passed in, we may decide
        * to append the 50 byte input lob segment to the output, or not.
        *---------------------------------------------
        * Create the output locator, right in the return buffer.
        */

       rc = sqludf_create_locator(SQL_TYP_CLOB, &lob_output);
       /* Error and exit if unable to create locator */
       if (rc) {
          memcpy (sqlstate, "38901", 5);
          /* special sqlstate for this condition */
          goto exit;
       }
       /* Find out the size of the input LOB value */
       rc = sqludf_length(lob_input, &input_len) ;
       /* Error and exit if unable to find out length */
       if (rc) {
          memcpy (sqlstate, "38902", 5);
          /* special sqlstate for this condition */
```

```
         goto exit;
      }
      /* Loop to read next 50 bytes, and append to result if it meets
       * the criteria.
       */
      for (input_pos = 1; (input_pos < input_len); input_pos += 50) {
        /* Read the next 50 (or less) bytes of the input LOB value */
        rc = sqludf_substr(lob_input, input_pos, 50,
                           (unsigned char *) lob_buf, &input_rec) ;
        /* Error and exit if unable to read the segment */
        if (rc) {
           memcpy (sqlstate, "38903", 5);
           /* special sqlstate for this condition */
           goto exit;
        }
        /* apply the criteria for appending this segment to result
  * if (...predicate involving buffer and criteria...) {
  * The example shows if the segment matches the first 6
         * characters with the criteria it is appended.
  */
        if (memcmp(lob_buf,criteria,6) == 0) {
           rc = sqludf_append(lob_output,
                      (unsigned char *) lob_buf, input_rec, &output_rec) ;
           /* Error and exit if unable to read the 50 byte segment */
           if (rc) {
              memcpy (sqlstate, "38904", 5);
              /* special sqlstate for this condition */
              goto exit;
    }
 }
        /* } end if criteria for inclusion met */
      } /* end of for loop, processing 50-byte chunks of input LOB
         * if we fall out of for loop, we are successful, and done.
  */
      *out_nul = 0;
      exit: /* used for errors, which will override null-ness of output. */
      return;
      }
```

Referring to this UDF code, observe that:

- There are includes for *sql.h*, where the type `SQL_TYP_CLOB` used in the `sqludf_create_locator()` call is defined, and `sqludf.h`, where the type `udf_locator` is defined.
- The first input argument, and the third input argument (which represents the function output) are defined as pointers to `sqludf_locator`, that is, they represent CREATE FUNCTION specifications of AS LOCATOR.
- The UDF does not test whether either input argument is null, as NOT NULL CALL is specified in the CREATE FUNCTION statement.
- In the event of error, the UDF exits with `sqlstate` set to 38*xxx*. This is sufficient to stop the execution of the statement referencing the UDF. The actual 38*xxx* SQLSTATE values you choose are not important to DB2, but can serve to differentiate the exception conditions which your UDF may encounter.
- By using the `input_rec` variable as the length of the data appended, the UDF takes care of any partial buffer condition.

Following is the CREATE FUNCTION statement for this UDF:

```
CREATE FUNCTION carve(CLOB(50M) AS LOCATOR, VARCHAR(255) )
  RETURNS CLOB(50M) AS LOCATOR
  NOT NULL CALL
  DETERMINISTIC
  NO SQL
```

```
        NO EXTERNAL ACTION
        LANGUAGE C
        PARAMETER STYLE DB2SQL
        EXTERNAL NAME 'MYLIB/LOBUDFS(lob_subsetter)' ;
```

Referring to this statement, observe that:

- NOT NULL CALL is specified, so the UDF will not be called if any of its input SQL arguments are NULL, and does not have to check for this condition.
- The function is specified as DETERMINISTIC, meaning that with a given input CLOB value and a given set of criteria, the result will be the same every time.

Now you can successfully run the following statement:

```
   strcpy(hvchar,"return this text 1                                "
                 "remove 1                                          "
                 "return this text 2                                "
                 "remove 2                                          ");
   exec sql set :hvloc = clob(:hvchar);
   exec sql set :hvloc2 = carve(:hvloc,'return');
   strcpy(hvchar,"");
   exec sql set :hvchar = char(:hvloc2);
```

The UDF is used to subset the value represented by the host variable :hvchar. The first and third 50 byte character segments are returned from the UDF.

API introduced: V5R3

Top | Database and File APIs |APIs by category

sqludf_create_locator_with_ccsid()—SQL LOB Create Locator With CCSID

Syntax

```
#include <sql.h>
#include <sqludf.h>

extern int SQL_API_FN sqludf_create_locator_with_ccsid(
   int          loc_type,
   long         ccsid,
   udf_locator ** loc_p)
```

Service Program Name: QSYS/QSQAPIS

Default Public Authority: *USE

Threadsafe: Yes

The **sqludf_create_locator_with_ccsid()** function creates a LOB locator with a given CCSID.

Parameters

loc_type

(Input) Type of LOB the locator represents. Valid locator types can be any of the types from sql.h representing LOBs. For example:

Type Name	Type Value	Description
SQL_TYP_BLOB	404	BLOB locator
SQL_TYP_NBLOB	405	BLOB locator that allows a null value
SQL_TYP_CLOB	408	CLOB locator
SQL_TYP_NCLOB	409	CLOB locator that allows a null value
SQL_TYP_DBCLOB	412	DBCLOB locator
SQL_TYP_NDBCLOB	413	DBCLOB locator that allows a null value

ccsid (Input) The CCSID of the data type the locator represents.

loc_p (Input/Output) Pointer to a pointer where the locator value is to be returned.

Authorities

No authorization is required.

Return Value

sqludf_create_locator_with_ccsid()

returns an integer. Possible values are:

0 **sqludf_create_locator_with_ccsid()** was successful. The information is returned in the buffer pointed to by *return_len_p*.

-3 **sqludf_create_locator_with_ccsid()** was not successful. An invalid parameter was passed into the function.

-429 **sqludf_create_locator_with_ccsid()** was not successful. The maximum number of concurrent LOB locators has been reached.

-901 **sqludf_create_locator_with_ccsid()** was not successful. An SQL system error has occurred.

-7034 **sqludf_create_locator_with_ccsid()** was not successful. LOB locators are not allowed with COMMIT(*NONE).

Error Messages

Message ID	Error Message Text
SQL7034 D	LOB locators are not allowed with COMMIT(*NONE).
SQL0901 D	SQL system error.
SQL0952 D	Processing of the SQL statement ended.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Usage Notes

1. This API is used to create a locator with a given CCSID. A LOB locator is only a mechanism used to refer to a LOB value during a transaction; it does not persist beyond the transaction in which it was created unless it is used with the HOLD LOCATOR statement.

Related Information

- "sqludf_append()—SQL LOB Append to Locator" on page 502—SQL LOB append locator
- "sqludf_create_locator()—SQL LOB Create Locator" on page 505—SQL LOB create locator
- "sqludf_free_locator()—SQL LOB Free Locator" on page 513—SQL LOB free locator
- "sqludf_length()—SQL LOB locator length" on page 515—SQL LOB locator length
- "sqludf_substr()—SQL LOB Substring Locator" on page 519—SQL LOB substring locator

Example

See Code disclaimer information for information pertaining to code examples.

This UDF takes a locator for an input LOB, and returns a locator for another LOB which is a subset of the input LOB. There are some criteria passed as a second input value, which tell the UDF how exactly to break up the input LOB.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sql.h>
#include <sqludf.h>

void SQL_API_FN lob_subsetter(
          udf_locator * lob_input,   /* locator of LOB value to carve up */
          char        * criteria,    /* criteria for carving */
          udf_locator * lob_output,  /* locator of result LOB value */
          sqlint16 * inp_nul,
          sqlint16 * cri_nul,
          sqlint16 * out_nul,
          char     * sqlstate,
          char     * funcname,
          char     * specname,
          char     * msgtext ) {

       /* local vars */
       short j;             /* local indexing var */
       int   rc;            /* return code variable for API calls */
       sqlint32  input_len;  /* receiver for input LOB length */
       sqlint32  input_pos;  /* current position for scanning input LOB */
       char lob_buf[100];    /* data buffer */
       sqlint32  input_rec;  /* number of bytes read by sqludf_substr */
       sqlint32  output_rec; /* number of bytes written by sqludf_append */
       long ccsid;           /* ccsid used to create locator */

       /*---------------------------------------------
        * UDF Program Logic Starts Here
        *---------------------------------------------
        * What we do is create an output handle, and then
        * loop over the input, 50 bytes at a time.
        * Depending on the "criteria" passed in, we may decide
        * to append the 50 byte input lob segment to the output, or not.
        *---------------------------------------------
        * Create the output locator, right in the return buffer.
        */

       ccsid = 37;
       rc = sqludf_create_locator_with_ccsid(SQL_TYP_CLOB, ccsid, &lob_output);
       /* Error and exit if unable to create locator */
       if (rc) {
          memcpy (sqlstate, "38901", 5);
          /* special sqlstate for this condition */
          goto exit;
       }
       /* Find out the size of the input LOB value */
       rc = sqludf_length(lob_input, &input_len) ;
       /* Error and exit if unable to find out length */
       if (rc) {
          memcpy (sqlstate, "38902", 5);
          /* special sqlstate for this condition */
          goto exit;
       }
       /* Loop to read next 50 bytes, and append to result if it meets
        * the criteria.
        */
       for (input_pos = 1; (input_pos < input_len); input_pos += 50) {
         /* Read the next 50 (or less) bytes of the input LOB value */
         rc = sqludf_substr(lob_input, input_pos, 50,
                            (unsigned char *) lob_buf, &input_rec) ;
         /* Error and exit if unable to read the segment */
         if (rc) {
            memcpy (sqlstate, "38903", 5);
            /* special sqlstate for this condition */
            goto exit;
         }
         /* apply the criteria for appending this segment to result
   * if (...predicate involving buffer and criteria...) {
   * The example shows if the segment matches the first 6
          * characters with the criteria it is appended.
   */
         if (memcmp(lob_buf,criteria,6) == 0) {
            rc = sqludf_append(lob_output,
                       (unsigned char *) lob_buf, input_rec, &output_rec) ;
            /* Error and exit if unable to read the 50 byte segment */
            if (rc) {
               memcpy (sqlstate, "38904", 5);
               /* special sqlstate for this condition */
               goto exit;
     }
  }
         /* } end if criteria for inclusion met */
       } /* end of for loop, processing 50-byte chunks of input LOB
          * if we fall out of for loop, we are successful, and done.
   */
       *out_nul = 0;
       exit: /* used for errors, which will override null-ness of output. */
       return;
       }
```

Referring to this UDF code, observe that:

- There are includes for *sql.h*, where the type `SQL_TYP_CLOB` used in the `sqludf_create_locator_with_ccsid()` call is defined, and `sqludf.h`, where the type `udf_locator` is defined.
- The first input argument, and the third input argument (which represents the function output) are defined as pointers to `sqludf_locator`, that is, they represent CREATE FUNCTION specifications of AS LOCATOR.
- The UDF does not test whether either input argument is null, as NOT NULL CALL is specified in the CREATE FUNCTION statement.
- In the event of error, the UDF exits with `sqlstate` set to 38*xxx*. This is sufficient to stop the execution of the statement referencing the UDF. The actual 38*xxx* SQLSTATE values you choose are not important to DB2, but can serve to differentiate the exception conditions which your UDF may encounter.
- By using the `input_rec` variable as the length of the data appended, the UDF takes care of any partial buffer condition.

Following is the CREATE FUNCTION statement for this UDF:

```
CREATE FUNCTION carve(CLOB(50M) AS LOCATOR, VARCHAR(255) )
  RETURNS CLOB(50M) AS LOCATOR
  NOT NULL CALL
  DETERMINISTIC
  NO SQL
  NO EXTERNAL ACTION
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  EXTERNAL NAME 'MYLIB/LOBUDFS(lob_subsetter)' ;
```

Referring to this statement, observe that:

- NOT NULL CALL is specified, so the UDF will not be called if any of its input SQL arguments are NULL, and does not have to check for this condition.
- The function is specified as DETERMINISTIC, meaning that with a given input CLOB value and a given set of criteria, the result will be the same every time.

Now you can successfully run the following statement:

```
strcpy(hvchar,"return this text 1                               "
              "remove 1                                         "
              "return this text 2                               "
              "remove 2                                         ");
exec sql set :hvloc = clob(:hvchar);
exec sql set :hvloc2 = carve(:hvloc,'return');
strcpy(hvchar,"");
exec sql set :hvchar = char(:hvloc2);
```

The UDF is used to subset the value represented by the host variable :hvchar. The first and third 50 byte character segments are returned from the UDF.

« API introduced: V5R3

Top | Database and File APIs | APIs by category

---

sqludf_free_locator()—SQL LOB Free Locator

Syntax

```
#include <sqludf.h>

extern int SQL_API_FN sqludf_free_locator(
   udf_locator * udfloc_p)
```

Service Program Name: QSYS/QSQAPIS

Default Public Authority: *USE

Threadsafe: Yes

The **sqludf_free_locator()** function frees a LOB locator.

Parameters

udfloc_p
(Input) Pointer to the LOB locator value.

Authorities

No authorization is required.

Return Value

sqludf_free_locator()

returns an integer. Possible values are:

0	**sqludf_free_locator()** was successful.
-3	**sqludf_free_locator()** was not successful. An invalid pointer was passed into the function.

-423	**sqludf_free_locator()** was not successful. The *udfloc_p* parameter points to an invalid locator value.
-901	**sqludf_free_locator()** was not successful. An SQL system error has occurred.
-7034	**sqludf_free_locator()** was not successful. LOB locators are not allowed with COMMIT(*NONE).

Error Messages

Message ID	**Error Message Text**
SQL7034 D	LOB locators are not allowed with COMMIT(*NONE).
SQL0901 D	SQL system error.
SQL0952 D	Processing of the SQL statement ended.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Usage Notes

1. This API is used to free a LOB locator.

Related Information

- "sqludf_append()—SQL LOB Append to Locator" on page 502—SQL LOB append locator
- "sqludf_create_locator()—SQL LOB Create Locator" on page 505—SQL LOB create locator
- "sqludf_create_locator_with_ccsid()—SQL LOB Create Locator With CCSID" on page 509—SQL LOB create locator with ccsid
- "sqludf_length()—SQL LOB locator length" on page 515—SQL LOB locator length
- "sqludf_substr()—SQL LOB Substring Locator" on page 519—SQL LOB substring locator

Example

See Code disclaimer information for information pertaining to code examples.

This example creates a CLOB locator and then frees the locator. The operations performed with the locator are left to the user to add.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sql.h>
#include <sqludf.h>

int main(int argc, char* argv[])
{
int   rc;                  /* return code variable for API calls */
udf_locator * udfloc_p;    /* pointer to locator */
udf_locator locator;       /* locator value */

udfloc_p = &locator;       /* set address to locator */
/* Create the locator */
rc = sqludf_create_locator(SQL_TYP_CLOB, &udfloc_p);
if (rc) {
  /* If create locator returned an error then return now */
  goto exit;
}

/* Perform operations using the locator */

/* Free the locator */
rc = sqludf_free_locator(udfloc_p);
if (rc) {
  /* If free locator returned an error then return now */
  goto exit;
}

exit: /* used for errors, which will override null-ness of output. */
return rc;
}
```

Referring to this code, observe that:

- There are includes for *sql.h*, where the type `SQL_TYP_CLOB` used in the `sqludf_create_locator()` call is defined, and `sqludf.h`, where the type `udf_locator` is defined.

This is an example of using the sqludf_free_locator API to free a locator.

API introduced: V5R3

Top | Database and File APIs | APIs by category

sqludf_length()—SQL LOB locator length

Syntax

```
#include <sqludf.h>

extern int SQL_API_FN sqludf_length(
   udf_locator * udfloc_p,
   long *        return_len_p)
```

Service Program Name: QSYS/QSQAPIS

Default Public Authority: *USE

Threadsafe: Yes

The **sqludf_length()** function returns the length of the LOB data represented by a LOB locator.

Parameters

udfloc_p
(Input) Pointer to the LOB locator value.

return_len_p
(Input/Output) Pointer to the length of the LOB data represented by the LOB locator.

Authorities

No authorization is required.

Return Value

sqludf_length()

returns an integer. Possible values are:

0	**sqludf_length()** was successful. The information is returned in the buffer pointed to by *return_len_p*.
-3	**squdf_length()** was not successful. An invalid parameter was passed into the function.
-423	**squdf_length()** was not successful. The *udfloc_p* parameter points to an invalid locator value.

-901	**sqludf_length()** was not successful. An SQL system error has occurred.
-7034	**sqludf_length()** was not successful. LOB locators are not allowed with COMMIT(*NONE).

Error Messages

Message ID	Error Message Text
SQL7034 D	LOB locators are not allowed with COMMIT(*NONE).
SQL0901 D	SQL system error.
SQL0952 D	Processing of the SQL statement ended.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Usage Notes

1. This API is used to find out the length of a LOB value when it receives a locator.

Related Information

- "sqludf_append()—SQL LOB Append to Locator" on page 502—SQL LOB append locator
- "sqludf_create_locator()—SQL LOB Create Locator" on page 505—SQL LOB create locator
- "sqludf_create_locator_with_ccsid()—SQL LOB Create Locator With CCSID" on page 509—SQL LOB create locator with ccsid
- "sqludf_free_locator()—SQL LOB Free Locator" on page 513—SQL LOB free locator
- "sqludf_substr()—SQL LOB Substring Locator" on page 519—SQL LOB substring locator

Example

See Code disclaimer information for information pertaining to code examples.

This UDF takes a locator for an input LOB, and returns a locator for another LOB which is a subset of the input LOB. There are some criteria passed as a second input value, which tell the UDF how exactly to break up the input LOB.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sql.h>
#include <sqludf.h>

void SQL_API_FN lob_subsetter(
          udf_locator * lob_input,   /* locator of LOB value to carve up */
          char        * criteria,    /* criteria for carving */
          udf_locator * lob_output,  /* locator of result LOB value */
          sqlint16 * inp_nul,
          sqlint16 * cri_nul,
          sqlint16 * out_nul,
          char     * sqlstate,
          char     * funcname,
          char     * specname,
          char     * msgtext ) {

       /* local vars */
       short j;             /* local indexing var */
       int   rc;            /* return code variable for API calls */
       sqlint32  input_len;  /* receiver for input LOB length */
       sqlint32  input_pos;  /* current position for scanning input LOB */
       char lob_buf[100];    /* data buffer */
       sqlint32  input_rec;  /* number of bytes read by sqludf_substr */
       sqlint32  output_rec; /* number of bytes written by sqludf_append */

       /*---------------------------------------------
```

```
         * UDF Program Logic Starts Here
         *-------------------------------------------
         * What we do is create an output handle, and then
         * loop over the input, 50 bytes at a time.
         * Depending on the "criteria" passed in, we may decide
         * to append the 50 byte input lob segment to the output, or not.
         *-------------------------------------------
         * Create the output locator, right in the return buffer.
         */

        rc = sqludf_create_locator(SQL_TYP_CLOB, &lob_output);
        /* Error and exit if unable to create locator */
        if (rc) {
           memcpy (sqlstate, "38901", 5);
           /* special sqlstate for this condition */
           goto exit;
        }
        /* Find out the size of the input LOB value */
        rc = sqludf_length(lob_input, &input_len) ;
        /* Error and exit if unable to find out length */
        if (rc) {
           memcpy (sqlstate, "38902", 5);
           /* special sqlstate for this condition */
           goto exit;
        }
        /* Loop to read next 50 bytes, and append to result if it meets
         * the criteria.
         */
        for (input_pos = 1; (input_pos < input_len); input_pos += 50) {
          /* Read the next 50 (or less) bytes of the input LOB value */
          rc = sqludf_substr(lob_input, input_pos, 50,
                             (unsigned char *) lob_buf, &input_rec) ;
          /* Error and exit if unable to read the segment */
          if (rc) {
             memcpy (sqlstate, "38903", 5);
             /* special sqlstate for this condition */
             goto exit;
          }
          /* apply the criteria for appending this segment to result
   * if (...predicate involving buffer and criteria...) {
   * The example shows if the segment matches the first 6
          * characters with the criteria it is appended.
   */
          if (memcmp(lob_buf,criteria,6) == 0) {
             rc = sqludf_append(lob_output,
                        (unsigned char *) lob_buf, input_rec, &output_rec) ;
             /* Error and exit if unable to read the 50 byte segment */
             if (rc) {
                memcpy (sqlstate, "38904", 5);
                /* special sqlstate for this condition */
                goto exit;
    }
  }
          /* } end if criteria for inclusion met */
        } /* end of for loop, processing 50-byte chunks of input LOB
           * if we fall out of for loop, we are successful, and done.
   */
        *out_nul = 0;
        exit: /* used for errors, which will override null-ness of output. */
        return;
        }
```

Referring to this UDF code, observe that:

- There are includes for *sql.h*, where the type `SQL_TYP_CLOB` used in the `sqludf_create_locator()` call is defined, and `sqludf.h`, where the type `udf_locator` is defined.

- The first input argument, and the third input argument (which represents the function output) are defined as pointers to sqludf_locator, that is, they represent CREATE FUNCTION specifications of AS LOCATOR.
- The UDF does not test whether either input argument is null, as NOT NULL CALL is specified in the CREATE FUNCTION statement.
- In the event of error, the UDF exits with sqlstate set to 38*xxx*. This is sufficient to stop the execution of the statement referencing the UDF. The actual 38*xxx* SQLSTATE values you choose are not important to DB2, but can serve to differentiate the exception conditions which your UDF may encounter.
- By using the input_rec variable as the length of the data appended, the UDF takes care of any partial buffer condition.

Following is the CREATE FUNCTION statement for this UDF:

```
CREATE FUNCTION carve(CLOB(50M) AS LOCATOR, VARCHAR(255) )
  RETURNS CLOB(50M) AS LOCATOR
  NOT NULL CALL
  DETERMINISTIC
  NO SQL
  NO EXTERNAL ACTION
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  EXTERNAL NAME 'MYLIB/LOBUDFS(lob_subsetter)' ;
```

Referring to this statement, observe that:

- NOT NULL CALL is specified, so the UDF will not be called if any of its input SQL arguments are NULL, and does not have to check for this condition.
- The function is specified as DETERMINISTIC, meaning that with a given input CLOB value and a given set of criteria, the result will be the same every time.

Now you can successfully run the following statement:

```
strcpy(hvchar,"return this text 1                              "
              "remove 1                                        "
              "return this text 2                              "
              "remove 2                                        ");
exec sql set :hvloc = clob(:hvchar);
exec sql set :hvloc2 = carve(:hvloc,'return');
strcpy(hvchar,"");
exec sql set :hvchar = char(:hvloc2);
```

The UDF is used to subset the value represented by the host variable :hvchar. The first and third 50 byte character segments are returned from the UDF.

« API introduced: V5R3

Top | Database and File APIs | APIs by category

sqludf_substr()—SQL LOB Substring Locator

Syntax

```
#include <sqludf.h>

extern int SQL_API_FN sqludf_substr(
   udf_locator * udfloc_p,
   long          start,
   long          length,
   unsigned char * buffer_p,
   long *        return_len_p)
```

Service Program Name: QSYS/QSQAPIS

Default Public Authority: *USE

Threadsafe: Yes

The **sqludf_substr()** function returns a substring of the LOB data the locator represents.

Parameters

udfloc_p
(Input) Pointer to the LOB locator value.

start (Input) The starting position of the substring. The first byte is byte 1.

length (Input) The number of bytes to return.

buffer_p
(Input) Pointer to the buffer where the substring is to be placed.

return_len_p
(Input/Output) Pointer to the number of bytes actually returned. This can be smaller then the length requested.

Authorities

No authorization is required.

Return Value

sqludf_substr()

returns an integer. Possible values are:

0	**sqludf_substr()** was successful. The information is returned in the buffer pointed to by *return_len_p*.
-3	**sqludf_substr()** was not successful. An invalid parameter was passed into the function.
-404	**sqludf_substr()** was not successful. The length of the substring is longer than the return buffer specified in *buffer_p*.
-423	**sqludf_substr()** was not successful. The *udfloc_p* parameter points to an invalid locator value.
-901	**sqludf_substr()** was not successful. An SQL system error has occurred.
-7034	**sqludf_substr()** was not successful. LOB locators are not allowed with COMMIT(*NONE).

Error Messages

Message ID	Error Message Text
SQL7034 D	LOB locators are not allowed with COMMIT(*NONE).
SQL0901 D	SQL system error.
SQL0952 D	Processing of the SQL statement ended.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Usage Notes

1. This API is used to see the bytes of the LOB value, when it has a locator.

Related Information

- "sqludf_append()—SQL LOB Append to Locator" on page 502—SQL LOB append to locator
- "sqludf_create_locator()—SQL LOB Create Locator" on page 505—SQL LOB create locator
- "sqludf_create_locator_with_ccsid()—SQL LOB Create Locator With CCSID" on page 509—SQL LOB create locator with ccsid
- "sqludf_free_locator()—SQL LOB Free Locator" on page 513—SQL LOB free locator
- "sqludf_length()—SQL LOB locator length" on page 515—SQL LOB locator length

Example

See Code disclaimer information for information pertaining to code examples.

This UDF takes a locator for an input LOB, and returns a locator for another LOB which is a subset of the input LOB. There are some criteria passed as a second input value, which tell the UDF how exactly to break up the input LOB.

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sql.h>
#include <sqludf.h>

void SQL_API_FN lob_subsetter(
          udf_locator * lob_input,   /* locator of LOB value to carve up */
          char        * criteria,    /* criteria for carving */
          udf_locator * lob_output,  /* locator of result LOB value */
          sqlint16 * inp_nul,
          sqlint16 * cri_nul,
          sqlint16 * out_nul,
          char     * sqlstate,
          char     * funcname,
          char     * specname,
          char     * msgtext ) {

       /* local vars */
       short j;             /* local indexing var */
       int   rc;            /* return code variable for API calls */
       sqlint32  input_len;  /* receiver for input LOB length */
       sqlint32  input_pos;  /* current position for scanning input LOB */
       char lob_buf[100];    /* data buffer */
       sqlint32  input_rec;  /* number of bytes read by sqludf_substr */
       sqlint32  output_rec; /* number of bytes written by sqludf_append */

       /*---------------------------------------------
        * UDF Program Logic Starts Here
        *---------------------------------------------
        * What we do is create an output handle, and then
        * loop over the input, 50 bytes at a time.
        * Depending on the "criteria" passed in, we may decide
        * to append the 50 byte input lob segment to the output, or not.
        *--------------------------------------------
        * Create the output locator, right in the return buffer.
        */

       rc = sqludf_create_locator(SQL_TYP_CLOB, &lob_output);
       /* Error and exit if unable to create locator */
       if (rc) {
          memcpy (sqlstate, "38901", 5);
          /* special sqlstate for this condition */
          goto exit;
       }
       /* Find out the size of the input LOB value */
       rc = sqludf_length(lob_input, &input_len) ;
       /* Error and exit if unable to find out length */
       if (rc) {
          memcpy (sqlstate, "38902", 5);
          /* special sqlstate for this condition */
          goto exit;
       }
       /* Loop to read next 50 bytes, and append to result if it meets
        * the criteria.
        */
       for (input_pos = 1; (input_pos < input_len); input_pos += 50) {
         /* Read the next 50 (or less) bytes of the input LOB value */
         rc = sqludf_substr(lob_input, input_pos, 50,
                            (unsigned char *) lob_buf, &input_rec) ;
         /* Error and exit if unable to read the segment */
         if (rc) {
            memcpy (sqlstate, "38903", 5);
            /* special sqlstate for this condition */
            goto exit;
         }
         /* apply the criteria for appending this segment to result
   * if (...predicate involving buffer and criteria...) {
   * The example shows if the segment matches the first 6
          * characters with the criteria it is appended.
   */
         if (memcmp(lob_buf,criteria,6) == 0) {
            rc = sqludf_append(lob_output,
                     (unsigned char *) lob_buf, input_rec, &output_rec) ;
            /* Error and exit if unable to read the 50 byte segment */
            if (rc) {
               memcpy (sqlstate, "38904", 5);
               /* special sqlstate for this condition */
               goto exit;
    }
  }
         /* } end if criteria for inclusion met */
       } /* end of for loop, processing 50-byte chunks of input LOB
          * if we fall out of for loop, we are successful, and done.
   */
       *out_nul = 0;
       exit: /* used for errors, which will override null-ness of output. */
       return;
       }
```

Referring to this UDF code, observe that:

- There are includes for *sql.h*, where the type `SQL_TYP_CLOB` used in the `sqludf_create_locator()` call is defined, and `sqludf.h`, where the type `udf_locator` is defined.
- The first input argument, and the third input argument (which represents the function output) are defined as pointers to `sqludf_locator`, that is, they represent CREATE FUNCTION specifications of AS LOCATOR.
- The UDF does not test whether either input argument is null, as NOT NULL CALL is specified in the CREATE FUNCTION statement.

- In the event of error, the UDF exits with `sqlstate` set to 38*xxx*. This is sufficient to stop the execution of the statement referencing the UDF. The actual 38*xxx* SQLSTATE values you choose are not important to DB2, but can serve to differentiate the exception conditions which your UDF may encounter.
- By using the `input_rec` variable as the length of the data appended, the UDF takes care of any partial buffer condition.

Following is the CREATE FUNCTION statement for this UDF:

```
CREATE FUNCTION carve(CLOB(50M) AS LOCATOR, VARCHAR(255) )
  RETURNS CLOB(50M) AS LOCATOR
  NOT NULL CALL
  DETERMINISTIC
  NO SQL
  NO EXTERNAL ACTION
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  EXTERNAL NAME 'MYLIB/LOBUDFS(lob_subsetter)' ;
```

Referring to this statement, observe that:

- NOT NULL CALL is specified, so the UDF will not be called if any of its input SQL arguments are NULL, and does not have to check for this condition.
- The function is specified as DETERMINISTIC, meaning that with a given input CLOB value and a given set of criteria, the result will be the same every time.

Now you can successfully run the following statement:

```
strcpy(hvchar,"return this text 1                                "
              "remove 1                                          "
              "return this text 2                                "
              "remove 2                                          ");
exec sql set :hvloc = clob(:hvchar);
exec sql set :hvloc2 = carve(:hvloc,'return');
strcpy(hvchar,"");
exec sql set :hvchar = char(:hvloc2);
```

The UDF is used to subset the value represented by the host variable :hvchar. The first and third 50 byte character segments are returned from the UDF.

« API introduced: V5R3

Top | Database and File APIs | APIs by category

Syntax Check SQL Statement (QSQCHKS) API

Required Parameter Group:

1	Source records containing SQL statement	
Input	Char(*)	
2	Record length	
Input	Binary(4)	
3	Number of records provided	
Input	Binary(4)	
4	Language	
Input	Char(10)	
5	Options	
Input	Char(*)	
6	Statement information	
Output	Char(*)	
7	Length of statement information	
Input	Binary(4)	
8	Number of records processed	
Output	Binary(4)	
9	Error code	
I/O	Char(*)	

Default Public Authority: *USE

Threadsafe: Yes

The Syntax Check SQL Statements (QSQCHKS) API calls the DB2 UDB for iSeries SQL parser to check the syntax of an SQL statement. If a specific language is specified, the parser will scan the source records passed according to the rules of the language. If a language is not passed, the parser will scan an SQL statement using the Interactive SQL syntax rules.

Authorities and Locks

No additional authority is required and no locks are acquired.

Required Parameters

Source records containing SQL statement
INPUT; CHAR(*)

The SQL statement that is to be parsed. This parameter can be passed as source text records for an HLL or as an SQL statement.

If the statement is contained in source text records for an HLL, the SQL statements must be in the form required by the precompiler for the specified language. For example, in COBOL, the statements must be preceded by EXEC SQL and followed by END-EXEC. Multiple statements will be processed. All the records will be processed as long as enough storage is provided for the statement information.

If a language is not specified, a single SQL statement must be passed without any additional delimiters (such as EXEC SQL or ;).

Record length
INPUT; BINARY(4)

The length of each record or the length of the SQL statement if language is *NONE. If language is *NONE the length must be between 1 and 32767. Record length for other languages must be at least as long as the right margin and cannot be longer than 100.

Number of records provided
INPUT; BINARY(4)

The number of source records to scan for the statement. This must be 1 if *NONE is specified for the language. If a language is specified, the number of records must be between 1 and 32767.

Language
INPUT; CHAR(10)

The programming language for which the syntax check is to be performed. Valid values include the following:

**NONE*	A syntax check is performed on the SQL statement using the Interactive SQL language syntax rules.
**CBL*	A syntax check is performed on the SQL statement using the COBOL language syntax rules.
**FTN*	A syntax check is performed on the SQL statement using the FORTRAN language syntax rules.
**PLI*	A syntax check is performed on the SQL statement using the PL/I language syntax rules.
**RPG*	A syntax check is performed on the SQL statement using the RPG language syntax rules.
**CLE*	A syntax check is performed on the SQL statement using the ILE C language syntax rules.
**CBLLE*	A syntax check is performed on the SQL statement using the ILE COBOL language syntax rules.
**RPGLE*	A syntax check is performed on the SQL statement using the ILE RPG language syntax rules.

Options
INPUT; CHAR(*)

The options required by SQL to parse the statement. The options must be specified as keys. The first part of the template is the number of keys passed, followed by variable length records for each option specified. For a description of the option data and keys, see “Format for Options” on page 525.

Number of options specified	BINARY(4) Total number of all the options (keys) specified. If this is 0, then defaults are used for the options.
Variable length option data	Variable length records containing the key indicating what key is passed, followed by the length of the data and the data.

Statement information
OUTPUT; CHAR(*)

The structure in which to return statement information for all statements processed. For the format of the structure, see “Statement Information” on page 527.

Length of area for statement information
Input; BINARY(4)

The length of the area in which to return statement information. This length must be at least 68 for information to be returned for statement. If a syntax error occurs, the length must be long enough to also contain the replacement text for the message. If more than 1 statement is processed, each statement after the first requires 44 bytes plus the length of the replacement text for any syntax errors.

Number of records processed
Output; BINARY(4)

The number of records processed. If the number of records processed is less than the number of records provided, the either an error occurred or there was not enough room in the statement information area to continue. This would never be greater than the number of records provided.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Format for Options

The following table defines the format for the options.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Number of keys
0	0	BINARY(4)	Key
04	04	BINARY(4)	Length of data
08	08	CHAR(*)	Data

If the length of character data is longer than the key field's data length, the data will be truncated at the right. No message will be issued.

If the length of character data is smaller than the key field's data length, the data will be padded with blanks at the right. No message will be issued.

If the same key is specified more than once, the last value for the option is used.

Field Descriptions

Data. The option used by SQL to scan the source and syntax check the SQL statement.

Key. Identifies a field of the options parameters. See "Keys" for the list of valid keys.

Length of data. The length of the data specified for the option.

Number of keys. The number of keys passed. This specifies the number of key arrays following this field. The arrays contain the key, length of data, and the data.

Keys

The following table lists the valid keys and the corresponding option.

Key	Type	Field
1	CHAR(10)	Naming convention
2	CHAR(1)	Operation
3	CHAR(1)	Character for delimited host strings
4	CHAR(1)	Character for delimited SQL strings
5	CHAR(1)	Character for the decimal point
6	BINARY(4)	left margin

Key	Type	Field
7	BINARY(4)	right margin
8	BINARY(4)	CCSID
9	CHAR(10)	Target release

Field Descriptions

CCSID. The CCSID to use for the source. The CCSID must be a valid CCSID. If not specified, the job CCSID will be used.

Character for delimited host strings. The character that is to be used to delimit host character strings. This parameter is not valid if the language is C or *NONE, and must be apostrophe if specified for FORTRAN, PL/I and RPG. If not specified for COBOL, the default is the quotation mark. Valid values include the following:

(')	apostrophe
(")	quotation mark

Character for delimited SQL strings. The character that is to be used to delimit character constants within an SQL statement. If the language is COBOL, either values can be specified and the default is quotation mark. If *NONE is specified for the language, either values can be specified and the default is apostrophe. For other languages, only the apostrophe can be specified. Valid values include the following:

(')	apostrophe
(")	quotation mark

Character for the decimal point. The character that is to be used for the decimal point. This parameter is valid for all languages. If not specified, the system value (QDECFMT) will be used. Valid values include the following:

(.)	period
(,)	comma

left margin. The left margin for the source. This parameter is only valid if language is PL/I or C and the valid values are from 1 to 80. If not specified, the default for PL/I is 2 and the default for C is 1. The left margin for RPG, COBOL, and FORTRAN is defined by the language and cannot be modified.

Naming convention. The naming convention used to qualify table names in the SQL statement. If this parameter is not passed, the default is *NONE. Valid values include the following:

**NONE*	The naming convention is not known. Errors in the qualification of table names are not returned.
**SYS*	Table names are qualified using the system naming convention in the form library/table.
**SQL*	Table names are qualified in the SQL naming convention in the form library.table.

Operation. The operation indicates what operations are to be performed by SQL. For performance, work areas can be reused across calls to the syntax checker, but SQL must be called eventually to terminate. The default is to syntax check the statement and terminate (2). However, for performance it is recommended that operation 0 be used in most cases when more than 1 SQL statement is to be checked. In this case, SQL must be called eventually to terminate. Valid values include the following:

0	Syntax check the statement and do not terminate. If this is specified, SQL must be called again to indicate the syntax check is complete.

1	Syntax check is complete. This option must be used to inform SQL to terminate when no more SQL statements need to be syntax checked.
2	Syntax check the statement and terminate.

Right margin. The right margin for the source. This parameter is only valid if language is PL/I or C and the valid values are from 1 to 80. The right margin must always be greater than the left margin. If not specified, the default for both PL/I and C is 80. The right margin for RPG, COBOL, and FORTRAN is defined by the language and cannot be modified.

Target release. The target release for which the statement should be syntax checked. If the statement cannot be taken back to the release specified, SQL7906 will be returned in the statement information. The default is the current release. Valid values include the following:

V2R3M0	The target release is Version 2, Release 3, Modification 0.
V3R0M5	The target release is Version 3, Release 0, Modification 5.
V3R1M0	The target release is Version 3, Release 1, Modification 0.
V3R6M0	The target release is Version 3, Release 6, Modification 0.

Statement Information

Offset			
Dec	**Hex**	**Type**	**Field**
Statement information header			
0	0	CHAR(10)	Message file name
10	0A	CHAR(10)	Message file library name
20	14	BINARY(4)	Number of statements processed
Statement information returned for statements processed (repeated for each statement processed for HLL):			
0	0	BINARY(4)	Length of information returned for this statement
4	4	BINARY(4)	Record number of first byte of statement
8	8	BINARY(4)	Column number of first byte of statement
12	C	BINARY(4)	Record number of last byte of statement
16	10	BINARY(4)	Column number of last byte of statement
20	14	BINARY(4)	Record number of the syntax error
24	18	BINARY(4)	Column number of the syntax error
28	1C	CHAR(7)	SQL message ID
35	23	CHAR(5)	SQLSTATE
40	28	BINARY(4)	Length of message replacement text
44	2C	CHAR(*)	Message replacement text

Field Descriptions

Column number of first byte of statement. The column containing the first byte of the beginning delimiter for the SQL statement. This would be the EXEC SQL in COBOL. This is blank if language is *NONE.

Column number of last byte of statement. The column containing the last byte of the ending delimiter for the SQL statement. This would be the END-EXEC in COBOL. If the record and column number of the first byte of the statement is set and the record and column number of the last byte of the statement is not, then we were processing a statement but did not find the end. No more records would be processed. This is blank if language is *NONE.

Column number of the syntax error. The column containing the syntax error if one was found.

Length of information returned for this statement. The length of the information returned for a single statement. This can be used as a displacement to the next statement.

Length of message replacement text. The length of the replacement text associated with the SQL message ID. If this is 0, then there is no replacement text for the message.

Message file library name. The library containing the SQL message file.

Message file name. The SQL Message file containing the message for the syntax error returned.

Message replacement text. A The replacement text for the message.

Number of statements processed. The number of statements processed. If called with language *NONE, this would always be 1 if enough space was provided for the statement information area.

Record number of first byte of statement. The record containing the first byte beginning delimiter for the SQL statement. This would be the EXEC SQL in COBOL. This is blank if language is *NONE.

Record number of last byte of statement. The record containing the last byte of the ending delimiter for the SQL statement. This would be the END-EXEC in COBOL. This is blank if language is *NONE.

Record number of the syntax error. The record containing the syntax error if one was found. If this is 0, then no error was found. If an error is found when language is *NONE, this value would be 1.

SQL message ID. If an error or warning is found, the message ID is set to th name of the message corresponding to the syntax error that occurred.

SQLSTATE. The SQLSTATE is additional information corresponding to the SQL return code. The SQLSTATEs are common across IBM SQL products for errors. For detailed information on this, see the DB2 UDB for SQL Programming Concepts topic.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
SQL0901 E	Record length parameter not valid.
SQL5502 E	Number of source records not valid.
SQL5503 E	Character for delimited host string not valid.
SQL5504 E	Character for delimited SQL string not valid.
SQL5505 E	Language not valid.

Message ID	Error Message Text
SQL5506 E	Naming convention not valid.
SQL5507 E	Margins not valid.
SQL5508 E	CCSID not valid.
SQL5509 E	Character specified as decimal point not valid.
SQL5510 E	Option parameter not valid.
SQL5511 E	Key field &1 not valid.
SQL5512 E	Number of keys not valid.
SQL5513 E	Target release not valid.
SQL5514 E	Length of data for key &1 not valid.
SQL5515 E	Length of area for statement information not valid.

API introduced: V3R1

Top | "Database and File APIs," on page 1 | APIs by category

Exit Programs

These are the Exit Programs for this category.

CLI Connection Exit Program

Required Parameter Group:

1	Connection user profile
Input	CHAR(10)

QSYSINC Member Name: None

Exit Point Name: QIBM_QSQ_CLI_CONNECT

Exit Point Format Name: CLIC0100

The CLI Connection exit program is called by CLI through the registration facility before the conection is made to the relational database. CLI must be running in server mode for the exit program to be called. The exit point supports one CLI Connection exit program at a time. This exit program can be used for such things as changing the library list depending on the user making the connection or to enable debug for the prestart job handling the SQL requests.

Authorities and Locks

You must have *ALLOBJ and *SECADM special authorities to register an exit program for the QIBM_QSQ_CLI_CONNECT exit point.

Required Parameter Group

Connection user profile
INPUT; CHAR(10)

The user profile that requested the connection.

Exit program introduced: V4R5

Top | “Database and File APIs,” on page 1 | APIs by category

Close Database File Exit Program

Required Parameter Group:	
1	Database close exit information
Input	Char(*)
Exit Point Name: QIBM_QDB_CLOSE	
Exit Point Format Name: DBCL0100	

The Close Database File exit program is called when a job is trying to lock a file that is currently held by another job due to existence of pseudo closed cursors. If the file is no longer being used, locks are being held over the file to improve the performance of the next SQL open in the job. The locks can be freed to allow the requesting job to use the file.

After a pseudo closed SQL cursor is fully closed, if an exit program is registered to this exit point, database will call the registered exit program in the job that had held the lock. The intent is that the job that had previously held the lock can now free up any additional resources it may have been holding related to the SQL cursor.

For information about adding an exit program to an exit point, see the Registration Facility APIs.

Note: The Close database file exit point ignores any return codes or error messages that are sent from the exit program.

Authorities and Locks

User Profile Authority
: *ALLOBJ and *SECADM to add or remove exit programs to the registration facility

Required Parameter

Database close information
: Input; CHAR(*)

  Information needed by the exit program for the database file to close.

Format of Database Close Exit Information

The following table shows the structure of the close database exit information for format DBCL0100. For a description of the fields in this format, see the Field Descriptions immediately following the table.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Database file name
10	0A	CHAR(10)	Database file library name

Offset			
Dec	Hex	Type	Field
20	14	CHAR(10)	Database file member name

Field Descriptions

Database file name. The database file name that another process is attempting to lock. This is always the 10-character system name.

Database file library name. The name of the library containing the database file.

Database file member name. The specific member that another process is attempting to lock. If the member name is *NONE, a file lock is being requested.

»

Usage Notes

Registration considerations.

Any exit program registered for this exit point must be threadsafe if it will be called in a job that has multiple threads.

« Exit program introduced: V5R1

Top | "Database and File APIs," on page 1 | APIs by category

Open Database File Exit Program

Required Parameter Group:

1	Open Database File Input Information
Input	Char(*)
2	Return Code
Output	Binary(4)

Exit Point Name: QIBM_QDB_OPEN

Exit Point Format Name: DBOP0100

QSYSINC Member Name: EDBOPNDB

The Open Database File exit program is called when a job is opening a database file. This exit is called in the job that is attempting to open the file. The exit program is passed a list of files referenced in the open request and the open options. The exit program may set a return code value to end the open request. When an open request is issued, the operating system calls the user-written exit program through the registration facility. For information about adding an exit program to an exit point, see the Registration Facility APIs.

If the file being opened is a logical file or a query, multiple files may be passed to the exit program. The originally requested files will be passed in as well as any underlying physical files. Only full opens will call the exit program. Hence,

- If the file is being opened as the result of an SQL statement, pseudo opens will not call the exit program.
- Shared opens will not call the exit program.

The Open Database File Exit Program can only be used with database objects. An open of a DDM file will not call the exit program on the source system, but it will call the exit program on the target system.

Authorities and Locks

User Profile Authority
*ALLOBJ and *SECADM to add or remove exit programs to the registration facility

Required Parameter Group

Open Database File Input Information
INPUT;CHAR(*)

Information needed by the exit program for the database files involved in the open. For the format of this parameter, see "DBOP0100 Format."

Open Database File Output Information
OUTPUT;BINARY(4)

Return code. The return code to indicate whether the open should be canceled. The valid values are:

0	The open should be rejected. Any remaining exit programs will not be called.
1	The open request should be accepted. The next exit program will be called or the open request will continue if there are no other exit programs. This is the default action.

DBOP0100 Format

The following tables show the format of the input information parameter for the exit program. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 533.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Size of Header for DBOP0100
4	4	CHAR(8)	Format Name
12	C	BINARY(4)	Offset to the referenced file array
16	10	BINARY(4)	Number of files in the referenced file array
20	14	BINARY(4)	Length of the referenced file array element
24	18	CHAR(10)	Job Name
34	22	CHAR(10)	User Name
44	2C	CHAR(6)	Job Number
50	32	CHAR(10)	Current User Name
60	3C	CHAR(1)	Database Query Open
61	3D	CHAR(*)	Reserved

The following structure shows the format of each array element of the Referenced File Array:

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Database file name
10	A	CHAR(10)	Database file library name
20	14	CHAR(10)	Database file member name
30	1E	CHAR(2)	Reserved
32	20	BINARY(4)	Database file type
36	24	BINARY(4)	Database open underlying physical file
40	28	CHAR(1)	Database open input option
41	29	CHAR(1)	Database open output option
42	2A	CHAR(1)	Database open update option
43	2B	CHAR(1)	Database open delete option
44	2C	CHAR(*)	Reserved

Field Descriptions

Current user name. The current user profile opening the database file.

Database file library name. The database file library name that is referenced in the open request.

Database file member name. The database file member name that is referenced in the open request. When processing partition tables the member will be returned as *ALL.

Database file name. A database file name that is referenced in the open request. This is always the 10-character system name.

Database file type. The type of the database file.

0	Physical database file.
1	Logical database file.

Database open delete option. The delete option specified for the file on the open request.

0	The file is not being opened for delete operations.
1	The file is being opened for delete operations.

Database open input option. The input option specified for the file on the open request.

0	The file is not being opened for input (read) operations.
1	The file is being opened for input (read) operations.

Database open output option. The output option specified for the file on the open request.

0	The file is not being opened for output (insert) operations.
1	The file is being opened for output (insert) operations.

Database open underlying physical file. The physical file underlying a logical file or view that was referenced in the open request.

0	The file was referenced in the open request directly or indirectly through an alias.

1	The file is an underlying physical file of a logical file or view that was referenced in the open request. This file was not directly referenced in the open request.

Database open update option. The update option specified for the file on the open request.

0	The file is not being opened for update operations.
1	The file is being opened for update operations.

Database query open. The files are being opened for a database query operation.

0	The file is not being opened for a database query.
1	The file is being opened for a database query.

Format Name. The name of the format being used.

Job Name. The name of the job issuing the open request.

Job Number. The number of the job issuing the open request.

Length of referenced file array element. The length of each element in the referenced file array.

Number of files in the referenced file array. The number of elements in the referenced file array.

Offset to the referenced file array. Indicates the offset from the start of the Open Database File Input Information to an array of files referenced in the open request.

Reserved. A reserved field.

Size of Header for DBOP0100. Size of header information.

User Name. The user name portion of the job issuing the open request.

Usage Notes

- If an exit program is being used for security reasons, it may want to ignore any referenced file array elements with a database open underlying physical file value of 1 since the user did not directly reference the underlying physical file. For example, the user may have authority to directly access the logical file, but not the underlying physical file.
- Exit program(s) will be called for user open file requests.
- If an exit program fails for any reason (not found, not authorized, function check in the program) the messages will be left in the joblog, but processing will continue.
- Exit program(s) will not be called for temporary files created by the system during query processing.
- Exit program(s) will not be called for files in the following system libraries(where ’xxxxx’ is the number of a primary auxiliary storage pool (ASP) and ’nnnn’ is the number of a basic user ASP.):
  - QTEMP
  - QSYS or QSYSxxxxx
  - QSYS2 or QSYS2xxxxx
  - SYSIBM or SYSIBxxxxx
  - QRCL or QRCYxxxxx
  - QRECOVERY or QRCYxxxxx
  - QRPLOBJ or QRPLxxxxx
  - QSPL or QSPLnnnn
- If an open request is issued for an MQT (Materialized Query Table), the MQT will be returned, not the files used to create it.

- In the case of multi-dataspace logical files, each underlying file will be returned.
- Exit program(s) registered for this exit point must be threadsafe and compiled with ACTGRP(*CALLER) because the exit program may be called as the result of an open file operation from an SQL External function. SQL external functions do not allow ACTGRP(*NEW).
- Exit program(s) will run in the job that issues the open request.
- Exit program(s) registered after a job has started might not be called for that existing job.
- Exit program(s) removed after a job has started may continue to be called for that existing job.
- Exit program(s) must be defined in the system ASP.
- When an exit program performs a file open or an SQL function, the open exit program will be called recursively. The exit program doing these operations must be coded to avoid recursion loops.

« Exit program introduced: V5R3

Top | "Database and File APIs," on page 1 | APIs by category

SQL Client Integration Exit Program

Required Parameter Group:

1	Interface level
Input	Binary(4)
2	Input format
Input	Char(*)
3	Length of input format
Input	Binary(4)
4	Input format name
Input	Char(8)
5	SQLCA
Output	Char(136)
6	CCSID
Output	Binary(4)
7	Output format
Output	Char(*)
8	Length of output format
Output	Binary(4)

QSYSINC Member Name: ERWSCI

The SQL Client Integration exit program enables SQL applications to access data managed by a database management system other than the OS/400 relational database. An application requester driver (ARD) program is the generic term for this type of exit program. The two terms are used interchangeably throughout this topic. The system calls the ARD program during the following operations:

- During the package creation step, performed by using the CRTSQLPKG or CRTSQLxxx commands, when the relational database (RDB) parameter matches the RDB name corresponding to the ARD program. During these calls, the system passes information about the SQL statements and host variables contained in the program. An ARD program can use this information to build what is comparable to an SQL access plan for the program. SQL statements passed to the ARD program at package creation can be correlated with the statement at run time by using the package name, collection, consistency token, and section number of the statement passed on the package creation and run-time interfaces.

- During CONNECT processing when the RDB name specified on the CONNECT statement matches the RDB name corresponding to the ARD program. During these calls information about the environment the statements are to run under is passed to the ARD program. An ARD program can use this information to establish the environment for running statements of the program.
- During processing of SQL statements when the current connection is to an RDB name corresponding to the ARD program. During these calls, the system passes information about the statement being run. An ARD program can either use this information alone or use the information in conjunction with the package name, collection, consistency token, and section number to process the SQL statement.

The CL commands that correspond to this exit program are the Add Relational Database Directory Entry (ADDRDBDIRE) and the Change Relational Database Directory Entry (CHGRBDDIRE) commands. Information about the ARD program must be defined to the system by adding it to the RDB directory using the ADDRDBDIRE command. Entries in the RDB directory that refer to ARD programs contain the keyword *ARDPGM in the remote location field. Each entry must identify the qualified ARD program name and the RDB name that it should be associated with. Also stored in the RDB directory entry is the level of interface that the ARD program expects to be called with. Currently the only value allowed is 1. An ARD program can be defined to process requests for several different RDBs by specifying the same ARD program for each RDB directory entry the ARD program is to process.

Restrictions

The following operations are not allowed in the ARD program or any program it calls:

- Commit operations for the commitment definition that is associated with the statement that the ARD program is currently handling.
- Rollback operations for the commitment definition that is associated with the statement that the ARD program is currently handling.
- End Commitment Control (ENDCMTCTL) command for the commitment definition that is associated with the statement that the ARD program is currently handling.
- Any SQL statements. SQL statements encountered while an ARD program is running will return messages SQLCODE (-84) and SQLSTATE (42612).

DECLARE CURSOR statements must parse successfully on the application requester to be used through this interface.

The ARD program must be in a library that is part of the system auxiliary storage pool (ASP number 1) or a configured basic ASP (ASP numbers 2-32).

The following functions are not supported for the SQL Client Integration exit program interface:

- Database large objects (BLOBs, CLOBs, DBCLOBs)
- Data links
- Passwords longer than ten characters
- Stored procedure result sets
- SQL statements longer than 32K
- Stored procedures with Commit on Return
- » Scrollable cursors
- Multi-row input
- Extended diagnostics
- User ids longer than 10 characters

- RDB aliases «

Required Parameter Group

Interface level
INPUT; BINARY(4)

The level of the ARD program. The only value that will currently be passed is 1 because no value can be specified on the RDB directory commands. It is possible that updates to the interface could be made in the future.

For example, such updates could include:
- Additional parameters.
- Changes to the input format structures.
- Changes to the SQLCA structure.
- Changes to the SQLDA structure.
- Changes to the output format structures.

Changes to enumerated values without changes to a structure will not result in a new interface level. Therefore, the ARD program should reject any unexpected values in the input format structures, or in the input SQLDA structure. In addition, the product identifier field on the ARCN0100 format can be used to determine the level of the local database that also identifies the values that could be expected for enumerated values in the input format and SQLDA structure. If updates are made to the interface, it may be possible for ARD programs to be registered with levels other than one by specifying the level on the RDB directory commands. At such time, a user registering the ARD program may incorrectly specify a level other than one for a program that only understands the level one interface. Therefore, ARD programs written to understand the current interface, the level one interface, should return an error if a level other than one is passed.

Input format
INPUT; CHAR(*);

The input format. The following table identifies the formats that the system will pass to the ARD program for each of the input format name values.

Note: General information on the nature of the functions associated with the various input formats listed may be found in the Distributed Relational Database Architecture Reference, SC26-4651. The chapter about the DRDA processing model and command flows should be of particular interest in this regard.

Relationship between Input Format Name and Input Format

Input Format Name	Input Format
ARCN0100	See "Format ARCN0100 (Connect Format)" on page 543.
ARDI0100	See "Format ARDI0100 (Disconnect Format)" on page 544.
ARBB0100	See "Format ARBB0100 (Begin Package Bind Format)" on page 545.
ARBS0100	See "Format ARBS0100 (Bind Statement for Package Creation Format)" on page 548.
AREB0100	See "Format AREB0100 (End of Package Bind Format)" on page 549.
ARPS0100	See "Formats ARPS0100 and ARPD0100 (Prepare Format)" on page 550.
ARPD0100	See "Formats ARPS0100 and ARPD0100 (Prepare Format)" on page 550.
ARXD0100	See "Formats ARXD0100 and ARXB0100 (Execute Bound Statement)" on page 552.
ARXB0100	See "Formats ARXD0100 and ARXB0100 (Execute Bound Statement)" on page 552.
ARXP0100	See "Format ARXP0100 (Execute Prepared Statement)" on page 555.

Input Format Name	Input Format
ARXI0100	See “Format ARXI0100 (Execute Immediate Statement Format)” on page 558.
AROC0100	See “Format AROC0100 (Open Cursor Format)” on page 560.
ARFC0100	See “Format ARFC0100 (Fetch from a Cursor Format)” on page 562.
ARCC0100	See “Format ARCC0100 (Close a Cursor Format)” on page 564.
ARDS0100	See “Format ARDS0100 (Describe a Statement Format)” on page 565.
ARDT0100	See “Format ARDT0100 (Describe Object Format)” on page 566.

Length of input format
INPUT; BINARY(4)

The length of the input format in bytes.

Input format name
INPUT; CHAR(8)

The format of the information passed to the ARD program. The possible format names follow:

ARCN0100
Connect format. This format will be used by the system to pass information to the ARD program when a user or application attempts to connect to an RDB name corresponding to the ARD program. This format will always be passed to the ARD program before any other formats for a given connection. If running under commitment control, the system will register an RDB resource with commitment control. However, the provider of the ARD program must also register with commitment control using the commitment control APIs to be informed of commit and rollback requests so that it processes those requests and closes cursors as necessary. Refer to “Commit APIs” on page 575 for a description of how to use commitment control APIs with ARD programs. See “Format ARCN0100 (Connect Format)” on page 543 for the structure of the input format that the system will pass to the ARD program for this input format name. See “Output Connect Format” on page 567 for the structure of the output format that the ARD program must return to the system in response to this input format name.

ARDI0100
Disconnect format. This format will be used by the system to pass information to the ARD program when a user, application, or the system attempts to disconnect from an RDB name that corresponds to the ARD program. See “Format ARDI0100 (Disconnect Format)” on page 544 for the structure of the input format that the system will pass to the ARD program for this input format name. No output format is returned from ARD program for this input format name.

ARBB0100
Begin package bind format. This format will be used by the system to pass information to the ARD program when a user or application attempts to create a package and specifies an RDB name corresponding to the ARD program. See “Format ARBB0100 (Begin Package Bind Format)” on page 545 for the structure of the input format that the system will pass to the ARD program for this input format name. No output format is returned from the ARD program for this input format name.

ARBS0100
Bind statement for package creation format. This format will be used by the system to pass information about an SQL statement to the ARD program when a user or application attempts to create a package and specifies an RDB name corresponding to the ARD program. See “Format ARBS0100 (Bind Statement for Package Creation Format)” on page 548 for the structure of the input format that the system will pass to the ARD program for this input format name. No output format is returned from ARD program for this input format name.

AREB0100
End of package bind format. This format will be used by the system to pass information to the ARD program about the end-the-package-creation function when a user or applications attempts to create a package and specifies an RDB name that corresponds to the ARD program. See "Format AREB0100 (End of Package Bind Format)" on page 549 for the structure of the input format that the system will pass to the ARD program for this input format name. No output format is returned from ARD program for this input format name.

ARPS0100
Prepare statement format. This format will be used by the system to pass information about an SQL statement to the ARD program when an application attempts to prepare a statement. See "Formats ARPS0100 and ARPD0100 (Prepare Format)" on page 550 for the structure of the input format that the system will pass to the ARD program for this input format name. No output format is returned from ARD program for this input format name.

ARPD0100
Prepare and describe format. This format will be used by the system to pass information about an SQL statement to the ARD program when an application attempts to prepare a statement and expects a description of the prepared statement to be returned into an SQL descriptor area (SQLDA). See "Formats ARPS0100 and ARPD0100 (Prepare Format)" on page 550 for the structure of the input format that the system will pass to the ARD program for this input format name. See "SQLDA" on page 573 for the structure of the output format that the ARD program must return to the system in response to this input format name.

ARXD0100
Execute bound statement that returns data format. This format will be used by the system to pass information about an SQL statement to the ARD program when an application attempts to execute a statement that expects data to be returned and was bound at package creation time by a call to the ARD program with format ARDSB001. An example of a statement that expects data to be returned is a SELECT INTO statement. See "Formats ARXD0100 and ARXB0100 (Execute Bound Statement)" on page 552 for the structure of the input format that the system will pass to the ARD program for this input format name. See "Output Execute Format" on page 568 for the structure of the output format that the ARD program must return to the system in response to this input format name.

ARXB0100
Execute bound statement that does not return data format. This format will be used by the system to pass information about an SQL statement to the ARD program when an application attempts to execute a statement that does not return data and was bound at package creation time by a call to the ARD program with format ARDSB001. See "Formats ARXD0100 and ARXB0100 (Execute Bound Statement)" on page 552 for the structure of the input format that the system will pass to the ARD program for this input format name. The output format that the ARD program must return to the system in response to this input format name is a character (CHAR(1)) field containing an indication of whether the statement resulted in an update. An update is any operation that results in a change to an object such that the object is under commitment control.

Valid values follow:

0	The operation did not result in an update.
1	An update occurred from the operation.

ARXP0100
Execute prepared statement format. This format will be used by the system to pass

information about an SQL statement to the ARD program when an application attempts to execute a statement that was previously prepared by a call to the ARD program with either format ARPS0100 or ARPD0100. See "Format ARXP0100 (Execute Prepared Statement)" on page 555 for the structure of the input format that the system will pass to the ARD program for this input format name. The output format that the ARD program must return to the system in response to this input format name is a character (CHAR(1)) field containing an indication of whether the statement resulted in an update. An update is any operation that results in a change to an object such that the object is under commitment control.

Valid values follow:

0	The operation did not result in an update.
1	An update occurred from the operation.

ARXI0100

Execute immediate statement format. This format will be used by the system to pass information about an SQL statement to the ARD program when an application attempts to execute a statement that was not previously prepared. See "Format ARXI0100 (Execute Immediate Statement Format)" on page 558 for the structure of the input format that the system will pass to the ARD program for this input format name. The output format that the ARD program must return to the system in response to this input format name is a character (CHAR(1)) field that contains an indication of whether the statement resulted in an update. An update is any operation that results in a change to an object such that the object is under commitment control.

Valid values follow:

0	The operation did not result in an update.
1	An update occurred from the operation.

AROC0100

Open a cursor format. This format will be used by the system to pass information about an SQL OPEN statement to the ARD program when an application attempts to execute the statement. See "Format AROC0100 (Open Cursor Format)" on page 560 for the structure of the input format that the system will pass to the ARD program for this input format name. See "Output Open Cursor Format" on page 569 for the structure of the output format that the ARD program must return to the system in response to this input format name.

ARFC0100

Fetch from a cursor format. This format will be used by the system to pass information about an SQL FETCH statement to the ARD program when an application attempts to execute the statement. See "Format ARFC0100 (Fetch from a Cursor Format)" on page 562 for the structure of the input format the system will pass to the ARD program for this input format name. See "Output Fetch Cursor Format" on page 570 for the structure of the output format that the ARD program must return to the system in response to this input format name.

ARCC0100

Close a cursor format. This format will be used by the system to pass information about an SQL CLOSE statement to the ARD program when an application attempts to execute the statement. See "Format ARCC0100 (Close a Cursor Format)" on page 564 for the structure of the input format that the system will pass to the ARD program for this input format name. No output format is returned from ARD program for this input format name.

ARDS0100
Describe an SQL statement format. This format will be used by the system to pass information about an SQL DESCRIBE STATEMENT statement to the ARD program when an application attempts to execute the statement. See "Format ARDS0100 (Describe a Statement Format)" on page 565 for the structure of the input format that the system will pass to the ARD program for this input format name. See "SQLDA" on page 573 for the structure of the output format that the ARD program must return to the system in response to this input format name.

ARDT0100
Describe an SQL table format. This format will be used by the system to pass information about an SQL DESCRIBE TABLE statement to the ARD program when an application attempts to execute the statement. See "Format ARDT0100 (Describe Object Format)" on page 566 for the structure of the input format that the system will pass to the ARD program for this input format name. See "SQLDA" on page 573 for the structure of the output format that the ARD program must return to the system in response to this input format name.

SQLCA
OUTPUT; CHAR(136)

The SQL communication area. This is used for returning diagnostic information. The format of the structure is standard, and can be included using the INCLUDE SQLCA statement in an SQL program. The SQLCA has the following fields (shown in the C-language format):

```
struct sqlca
 {
    unsigned char  sqlcaid[8];
    long           sqlcabc;
    long           sqlcode;
    short          sqlerrml;
    unsigned char  sqlerrmc[70];
    unsigned char  sqlerrp[8];
    long           sqlerrd[6];
    unsigned char  sqlwarn[11];
    unsigned char  sqlstate[5];
 };
```

Fields that must be set by the ARD program prior to returning follow:

sqlcaid	An eye-catcher for diagnostic purposes. This must be set to 'SQLCA'.
sqlcabc	The byte length of the SQLCA. This must be set to 136.
sqlcode	The SQL return code. If the sqlcode is 0, the statement completed successfully although a warning may have occurred. If the sqlcode is positive, the statement completed successfully but a warning occurred during execution. If the sqlcode is negative, an error occurred while running the statement. A discussion about setting the sqlcode to match a system message identifier follows this list of fields.
sqlerrp	The program that detected the error and built the SQLCA.
sqlerrd[2]	The number of rows affected for successful INSERT, and DELETE statements. This cannot be zero for INSERT, UPDATE, and DELETE statements when the sqlstate is 00000.
sqlstate	A return code field that indicates the outcome of the most recently executed SQL statement. An sqlstate of 00000 indicates an unqualified successful completion. ANS/ISO standard sqlstate values should be used and are documented in the DB2 UDB for iSeries SQL Programming topic in the Information center.

Another field in the SQLCA, sqlerrmc, is used to return additional pertinent information about the last statement run. Tokens in this field must be separated by X'FF' to be interpreted properly.

Each sqlcode has a corresponding message in message file QSQLMSGin library QSYS.For negative SQLCODEs and positive SQLCODEs other than +100, the corresponding message for the

SQLCODE will be put in the job log. In addition, messages about how a statement ran are also put in the job log when running in debug mode. An ARD program can determine if the application is running in debug mode by using the debug APIs. The message ID is constructed by appending the absolute value (5 digits) of the sqlcode to SQ and changing the third character to L if the third character is a zero. For example, if the sqlcode is -501, the message identifier is SQL0501. Each message may optionally have replacement variables. These variables are placed in the sqlerrmc field of the SQLCA in the previous paragraph. A Display Message Description (DSPMSGD) command or format RTVM0300 of the Retrieve Message (QMHRTVM) API can be used to determine the correct length and type for replacement variables for a particular message. The sqlerrmc field for a message should be built up according to the field data for that message. Refer to SQLCODEs and SQLSTATEs in the DB2 UDB for iSeries SQL Programming topic in the Information Center for more information about SQLCODEs, SQLSTATEs, and their meaning.

CCSID
OUTPUT; BINARY(4)

The CCSID of the sqlerrm, sqlerrp, and sqlwarn fields in the SQLCA.

Output format
OUTPUT; CHAR(*)

The format of the information passed from the ARD program. The following table identifies the formats that the ARD program must return for each of the input format name values that the system will pass to it.

Relationship between Input Format Name and Output Format

Input Format Name	Output Format
ARCN0100	See "Output Connect Format" on page 567.
ARDI0100	No output format.
ARBB0100	No output format.
ARBS0100	No output format.
AREB0100	No output format.
ARPS0100	No output format.
ARPD0100	See "SQLDA" on page 573.
ARXD0100	See "Output Execute Format" on page 568.
ARXB0100	Update performed (see the description following this table).
ARXP0100	Update performed (see the description following this table).
ARXI0100	Update performed (see the description following this table).
AROC0100	See "Output Open Cursor Format" on page 569.
ARFC0100	See "Output Fetch Cursor Format" on page 570.
ARCC0100	No output format.
ARDS0100	See "SQLDA" on page 573.
ARDT0100	See "SQLDA" on page 573.

Update performed. CHAR(1)

An indicator of whether the statement resulted in an update. An update is any operation that results in a change to an object such that the object is under commitment control. Valid values follow:

0	The operation did not result in an update.
1	Update occurred from the operation.

Length of output format
OUTPUT; BINARY(4)

The length of the output format in bytes. This must be zero for the following input format names: ARDI0100, ARBB0100, ARBS0100, AREB0100, ARPS0100, and ARCC0100.

Input Format Structures

In the following structures, the CCSID of the character fields is the job CCSID unless a specific CCSID field is included in the format for the field.

Format ARCN0100 (Connect Format)

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Output format buffer size
4	4	BINARY(4)	Activation group number
8	8	CHAR(18)	RDB name
26	1A	CHAR(10)	Device name
36	24	CHAR(8)	Mode name
44	2C	CHAR(8)	Remote location name
52	34	CHAR(8)	Local location name
60	3C	CHAR(8)	Remote network identifier
68	44	CHAR(8)	TPN name
76	4C	CHAR(10)	User ID
86	56	CHAR(10)	Password
96	60	CHAR(8)	Product identifier

Field Descriptions for Format ARCN0100

Activation group number. The activation group number of the program that is performing the request. See “Activation Group” on page 570 for a description of what an activation group is.

Device name. The device name that is specified in the directory entry. This will be blank if RMTLOCNAME(*ARDPGM) is specified for the RDB directory entry.

Local location name. The local location name that is specified in the directory entry. This will be blank if RMTLOCNAME(*ARDPGM) is specified for the RDB directory entry.

Mode name. The mode name that is specified in the directory entry. This will be blank if RMTLOCNAME(*ARDPGM) is specified for the RDB directory entry.

Output format buffer size. The amount of storage allocated for the output format that is returned by the ARD program. The length of the output format must be less than or equal to this value. It must also conform to the description of the output format associated with this input format. See the description of the output format parameter (page 542) for a description of the output format associated with this input format.

Password. The password that the application or user specified on the CONNECT statement. This field is blank if no password is specified. The system does not verify that this password is correct.

Product identifier. The product identifier for the local database in the form QSQ*vvrrm*, where:

- *vv* is a 2-digit version identifier such as 03.
- *rr* is a 2-digit release identifier such as 01.
- *m* is a 1-digit modification level such as 0.

For example, if the local database is Version 3 Release 1 Modification 0 of DB2/400, the product identifier is QSQ03010.

RDB name. The name of the relational database that the request was directed to.

Remote location name. The remote location name that is specified in the RDB directory entry. This will be *ARDPGM if RMTLOCNAME(*ARDPGM) is specified for the RDB directory entry.

Remote network identifier. The remote network identifier that is specified in the directory entry. This will be blank if RMTLOCNAME(*ARDPGM) is specified for the RDB directory entry.

TPN name. The transaction program name that is specified in the directory entry. This will be blank if RMTLOCNAME(*ARDPGM) is specified for the RDB directory entry.

User ID. The user identifier that the application or user specified on the CONNECT statement. This field is blank if no user ID is specified.

Format ARDI0100 (Disconnect Format)

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Output format buffer size
4	4	BINARY(4)	Activation group number
8	8	CHAR(18)	RDB name
26	1A	CHAR(2)	Reserved
28	1C	BINARY(4)	Disconnect type

Field Descriptions for Format ARDI0100

Activation group number. The activation group number of the program that is performing the request. See "Activation Group" on page 570 for a description of what an activation group is.

Disconnect type. The system will set this field to indicate the type of disconnection that is being performed. Values passed follow:

1	Disconnection is occurring because the application performed a DISCONNECT statement for only the relational database that the ARD program is associated with or the application was compiled with the RDB connection method of *RUW and it performed a CONNECT. If the ARD program returns a negative SQLCODE, the disconnection fails and the connection is not ended. The system will never pass this value to the ARD program if the ARD program indicated the conversation uses a protected conversation on the ARCN0100 format.
2	Disconnection is occurring because a DISCONNECT ALL was performed by the application or because all connections for the activation group are ending by an implicit disconnection. Regardless of the SQLCODE value returned by the ARD program, the connection will be ended. The system will never pass this value to the ARD program if the ARD program indicated that the conversation uses a protected conversation on the ARCN0100 format.

3	Disconnection is occurring as part of a commit or rollback. The connection is ending for one of the following reasons: • The connection was released and a commit is being performed. • The SQL application ended. • The activation group ended. • An error was detected during an earlier call to the ARD program. The connection had pending changes.
4	Disconnection is occurring because an error was detected during an earlier call to the ARD program. Regardless of the SQLCODE value returned by the ARD program, the connection will be ended.

Output format buffer size. The amount of storage allocated for the output format that is returned by theARD program. The length of the output format must be less than or equal to this value. It must also conform to the description of the output format associated with this input format. See the description of the output format parameter (page 542) for a description of the output format associated with this input format.

RDB name. The name of the relational database that the request was directed to.

Reserved. An ignored field.

Format ARBB0100 (Begin Package Bind Format)

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Output format buffer size
4	4	BINARY(4)	Activation group number
8	8	CHAR(18)	RDB name
26	1A	CHAR(18)	Package collection
44	2C	CHAR(18)	Package name
62	3E	CHAR(8)	Package consistency token
70	46	CHAR(2)	Reserved
72	48	BINARY(4)	CCSID
76	4C	CHAR(1)	Existence required
77	4D	CHAR(1)	Errors allowed
78	4E	CHAR(1)	Replace allowed
79	4F	CHAR(1)	String delimiter
80	50	CHAR(1)	Decimal delimiter
81	51	CHAR(1)	Blocking type
82	52	CHAR(10)	Date format
92	5C	CHAR(10)	Time format
102	66	CHAR(10)	Isolation level
112	70	CHAR(18)	Default collection
130	82	CHAR(50)	Text

Field Descriptions for Format ARBB0100

Activation group number. The activation group number of the program that is performing the request. See "Activation Group" on page 570 for a description of what an activation group is.

Blocking type. An indicator of when blocking should be performed for read-only cursors in the program. This value may be overridden on the call to the ARD program with the AROC0100 format when the cursor is opened. Values passed follow:

0	Blocking is never performed.
1	Blocking is only performed for cursors declared FOR FETCH ONLY or if there are no dynamic statements or positioned UPDATE or DELETE statements for the cursor.
2	Blocking is performed as long as the cursor is not declared FOR UPDATE and there are no positioned UPDATE or DELETE statements for the cursor.

CCSID. The CCSID of the text. This will always be set to 500.

Date format. The format that is used when the exit program accesses date result columns. Values passed follow (where *m*=month, *d*=day, and *y*=year):

**USA*	The United States date format *mm/dd/yyyy*.
**ISO*	The International Organization for Standardization (ISO) date format *yyyy-mm-dd*.
**EUR*	The European date format *dd.mm.yyyy*.
**JIS*	The Japanese Industrial Standard date format *yyyy-mm-dd*.

Decimal delimiter. The statement decimal delimiter for the SQL statements. Values passed follow:

.	The value used as the decimal point in numeric literals is a period.
,	The value used as the decimal point in numeric literals is a comma.

Default collection. The name of the collection identifier that is used for the unqualified names of the tables, views, indexes, and SQL packages. This parameter applies only to static SQL statements. A special value of *NONE indicates no default collection.

Errors allowed. Whether errors are allowed. Values passed follow:

0	All statements are checked for correct syntax and semantics. If any error occurs, the package should not be created. When processing the AREB0100 format and an error occurs, the ARD program should return a negative sqlcode for any statement in error. When the AREB0100 format is processed, the ARD program should return a negative sqlcode.
1	Even if errors occur while processing the statements, the package should be created. Reserved sections should be generated for statements in error. When processing the ARBS0100 format and an error occurs, the ARD program should return a negative sqlcode for any statement in error. However, when processing the AREB0100 format, the ARD program should return a non-negative sqlcode.

Existence required. Whether existence of and authority to an object is required. Values passed follow:

0	The absence of an object or lack of authority to an object is not treated as an error. When processing the ARBS0100 format and an object is not found or an authority error occurs, a non-negative sqlcode should be returned from the ARD program in the SQLCA parameter.
1	The absence of an object or lack of authority to an object is treated as an error. When processing the ARBS0100 format and an object is not found or an authority error occurs, a negative sqlcode should be returned from the ARD program in the SQLCA parameter.

Isolation level. The level of record locking that occurs under commitment control. Values passed follow:

**CHG*	The following are locked until the end of the unit of work (transaction): • Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements • Rows that are updated, deleted, and inserted Uncommitted changes in other jobs can be seen.
**CS*	The following are locked until the end of the unit of work (transaction): • Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements • Rows that are updated, deleted, and inserted A row that is selected but not updated is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.
**ALL*	The following are locked until the end of the unit of work (transaction): • Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements • Rows that are selected, updated, deleted, and inserted Uncommitted changes in other jobs cannot be seen.

Output format buffer size. The amount of storage allocated for the output format that is returned by the ARD program. The length of the output format must be less than or equal to this value. It must also conform to the description of the output format associated with this input format. See the description of the output format parameter (page 542) on for a description of the output format associated with this input format.

Package collection. The collection for the package that is being created. A collection is a name that provides a logical grouping for SQL objects.

Package consistency token. The consistency token for the package that is being created. Refer to "Consistency Token" on page 570 for a description of this field.

Package name. The name of the package that is being created.

RDB name. The name of the relational database that the request was directed to.

Replace allowed. Whether the package can be replaced. Values passed follow:

0	The SQL package is not created if the SQL package of the same name already exists in the specified collection. When processing the AREB0100 format and the package with the same name already exists, the ARD program returns a negative sqlcode.
1	The SQL package is created and any existing SQL package of the same name in the specified collection replaced. The authorities for the existing SQL package are kept for the new SQL package.

Reserved. An ignored field.

String delimiter. The statement string delimiter for the SQL statements. Values passed follow:

′	The character used as the string delimiter is the apostrophe (′).
" "	The character used as the string delimiter is the quotation mark (").

Text. Text that briefly describes the packages function.

Time format. The format that is used when the exit program accesses time result columns. Values passed follow (where *h*=hour, *m*=minute, and *s*=second):

**USA*	The United States time format *hh:mm xx* is used, where *xx* is A.M. or P.M.
**ISO*	The International Organization for Standardization (ISO) time format *hh.mm.ss*.
**EUR*	The European time format *hh.mm.ss*.
**JIS*	The Japanese Industrial Standard time format *hh:mm:ss*.

Format ARBS0100 (Bind Statement for Package Creation Format)

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Output format buffer size
4	4	BINARY(4)	Activation group number
8	8	CHAR(18)	RDB name
26	1A	CHAR(18)	Package collection
44	2C	CHAR(18)	Package name
62	3E	CHAR(8)	Package consistency token
70	46	CHAR(2)	Reserved
72	48	BINARY(4)	Section number
76	4C	BINARY(4)	CCSID
80	50	BINARY(4)	Offset to SQLDA
84	54	BINARY(4)	Length of SQLDA
88	58	BINARY(4)	Offset to SQL statement
92	5C	BINARY(4)	Length of SQL statement
		CHAR(*)	SQLDA
		CHAR(*)	SQL statement

Field Descriptions for Format ARBS0100

Activation group number. The activation group number of the program that is performing the request. See "Activation Group" on page 570 for a description of what an activation group is.

CCSID. The CCSID of the statement text.

Length of SQLDA. The length of the SQLDA structure that describes the host variables that are used on the statement. If zero, no host variables were used on the statement.

Length of SQL statement. The length of the SQL statement as contained in the program.

Output format buffer size. The amount of storage allocated for the output format that is returned by the ARD program. The length of the output format must be less than or equal to this value. It must also conform to the description of the output format associated with this input format. See the description of the output format parameter (page 542) on for a description of the output format associated with this input format.

Offset to SQLDA. The offset from the start of the input format structure to the SQLDA structure that describes the host variables that are used on the statement. If zero, no host variables were used on the statement.

Offset to SQL statement. The offset from the start of the input format structure to the SQL statement as contained in the program.

Package collection. The collection for the package being created. A collection is a name that provides a logical grouping for SQL objects.

Package consistency token. The consistency token for the package being created. Refer to “Consistency Token” on page 570 for a description of this field.

Package name. The name of the package being created.

RDB name. The name of the relational database that the request was directed to.

Reserved. An ignored field.

Section number. The section number of the statement. Refer to “Section Number” on page 571 for a description of this field.

SQLDA. An SQLDA structure that describes the host variables that are used on the statement. The SQLDA structure is described in “SQLDA” on page 573. The SQLDATA and SQLIND pointers are set to NULL for package creation.

SQL statement. The SQL statement as contained in the program except that `:H` has been substituted for the host variable identifiers.

Format AREB0100 (End of Package Bind Format)

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Output format buffer size
4	4	BINARY(4)	Activation group number
8	8	CHAR(18)	RDB name
26	1A	CHAR(18)	Package collection
44	2C	CHAR(18)	Package name
62	3E	CHAR(8)	Package consistency token
70	46	CHAR(2)	Reserved
72	48	BINARY(4)	Maximum section number

Field Descriptions for Format AREB0100

Activation group number. The activation group number of the program that is performing the request. See “Activation Group” on page 570 for a description of what an activation group is.

Maximum section number. The last section number in the package. This value may be greater than the last number passed on a call to the ARD program with format ARBS0100 when section numbers are reserved. Refer to “Section Number” on page 571 for more information on section numbers.

Output format buffer size. The amount of storage allocated for the output format that is returned by the ARD program. The length of the output format must be less than or equal to this value. It must also conform to the description of the output format associated with this input format. See the description of the output format parameter (page 542) on for a description of the output format associated with this input format.

Package collection. The collection for the package being created. A collection is a name that provides a logical grouping for SQL objects.

Package consistency token. The consistency token for the package being created. Refer to “Consistency Token” on page 570 for a description of this field.

Package name. The name of the package being created.

RDB name. The name of the relational database that the request was directed to.

Reserved. An ignored field.

Formats ARPS0100 and ARPD0100 (Prepare Format)

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Output format buffer size
4	4	BINARY(4)	Activation group number
8	8	CHAR(18)	RDB name
26	1A	CHAR(18)	Package collection
44	2C	CHAR(18)	Package name
62	3E	CHAR(8)	Package consistency token
70	46	CHAR(2)	Reserved
72	48	BINARY(4)	Section number
76	4C	BINARY(4)	CCSID
80	50	CHAR(1)	String delimiter
81	51	CHAR(1)	Decimal delimiter
82	52	CHAR(10)	Date format
92	5C	CHAR(10)	Time format
102	66	CHAR(10)	Isolation level
112	70	BINARY(4)	Offset to SQL statement
116	74	BINARY(4)	Length of SQL statement
120	78	CHAR(18)	Statement name
		CHAR(*)	SQL statement

Field Descriptions for Formats ARPS0100 and ARPD0100

Activation group number. The activation group number of the program that is performing the request. See “Activation Group” on page 570 for a description of what an activation group is.

CCSID. The CCSID of the statement text and statement name.

Date format. The format that is used when the exit program accesses date result columns. Values passed follow (where *m*=month, *d*=day, and *y*=year):

**USA*	The United States date format *mm/dd/yyyy*.
**ISO*	The International Organization for Standardization (ISO) date format *yyyy-mm-dd*.
**EUR*	The European date format *dd.mm.yyyy*.

**JIS*	The Japanese Industrial Standard date format *yyyy-mm-dd*.

Decimal delimiter. The statement decimal delimiter for the SQL statements. Values passed follow:

.	The value used as the decimal point in numeric literals is a period.
,	The value used as the decimal point in numeric literals is a comma.

Isolation level. The level of record locking that occurs under commitment control. Values passed follow:

**CHG*
: The following are locked until the end of the unit of work (transaction):

- Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements
- Rows that are updated, deleted, and inserted

Uncommitted changes in other jobs can be seen.

**CS*
: The following are locked until the end of the unit of work (transaction):

- Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements
- Rows that are updated, deleted, and inserted

A row that is selected but not updated is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

**ALL*
: The following are locked until the end of the unit of work (transaction):

- Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements
- Rows that are selected, updated, deleted, and inserted

Uncommitted changes in other jobs cannot be seen.

Length of SQL statement. The length of the statement string being prepared.

Offset to SQL statement. The offset from the start of the input format structure to the statement string being prepared.

Output format buffer size. The amount of storage allocated for the output format that is returned by the ARD program. The length of the output format must be less than or equal to this value. It must also conform to the description of the output format associated with this input format. See the description of the output format parameter (page 542) on for a description of the output format associated with this input format.

Package collection. The collection for the package that the statement is associated with. A collection is a name that provides a logical grouping for SQL objects.

Package consistency token. The consistency token for the package that the statement is associated with. Refer to "Consistency Token" on page 570 for a description of this field.

Package name. The name of the package that the statement is associated with.

RDB name. The name of the relational database that the request was directed to.

Reserved. An ignored field.

Section number. The section number that the statement is associated with. Refer to "Section Number" on page 571 for more information on section numbers.

SQL statement. The statement string being prepared.

Statement name. The SQL statement name that is specified on the PREPARE statement.

String delimiter. The statement string delimiter for the SQL statements. Values passed follow:

′	The character used as the string delimiter is the apostrophe (′).
″	The character used as the string delimiter is the quotation mark (“).

Time format. The format that is used when the exit program accesses time result columns. Values passed follow (where *h*=hour, *m*=minute, and *s*=second):

**USA*	The United States time format *hh:mm xx* is used, where *xx* is A.M. or P.M.
**ISO*	The International Organization for Standardization (ISO) time format *hh.mm.ss*.
**EUR*	The European time format *hh.mm.ss*.
**JIS*	The Japanese Industrial Standard time format *hh:mm:ss*.

Formats ARXD0100 and ARXB0100 (Execute Bound Statement)

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Output format buffer size
4	4	BINARY(4)	Activation group number
8	8	CHAR(18)	RDB name
26	1A	CHAR(18)	Package collection
44	2C	CHAR(18)	Package name
62	3E	CHAR(8)	Package consistency token
70	46	CHAR(2)	Reserved
72	48	BINARY(4)	Section number
76	4C	BINARY(4)	CCSID
80	50	CHAR(1)	String delimiter
81	51	CHAR(1)	Decimal delimiter
82	52	CHAR(10)	Date format
92	5C	CHAR(10)	Time format
102	66	CHAR(10)	Isolation level
112	70	CHAR(18)	Default collection
130	82	CHAR(2)	Reserved
132	84	BINARY(4)	Offset to input SQLDA
136	88	BINARY(4)	Length of input SQLDA
140	8C	BINARY(4)	Offset to SQL statement
144	90	BINARY(4)	Length of SQL statement
148	94	BINARY(4)	Offset to DECLARE PROCEDURE
152	98	BINARY(4)	Length of DECLARE PROCEDURE
156	9C	BINARY(4)	Offset to procedure name

Offset			
Dec	Hex	Type	Field
160	A0	BINARY(4)	Length of procedure name
		CHAR(*)	Input SQLDA
		CHAR(*)	SQL statement
		CHAR(*)	DECLARE PROCEDURE statement
		CHAR(*)	Procedure name

Field Descriptions for Formats ARXD0100 and ARXB0100

Activation group number. The activation group number of the program that is performing the request. See "Activation Group" on page 570 for a description of what an activation group is.

CCSID. The CCSID of statement text.

Date format. The format that is used when the exit program accesses date result columns. Values passed follow (where *m*=month, *d*=day, and *y*=year):

**USA*	The United States date format *mm/dd/yyyy*.
**ISO*	The International Organization for Standardization (ISO) date format *yyyy-mm-dd*.
**EUR*	The European date format *dd.mm.yyyy*.
**JIS*	The Japanese Industrial Standard date format *yyyy-mm-dd*.

Decimal delimiter. The statement decimal delimiter for the SQL statements. Values passed follow:

.	The value used as the decimal point in numeric literals is a period.
,	The value used as the decimal point in numeric literals is a comma.

DECLARE PROCEDURE statement. The DECLARE PROCEDURE statement as contained in the program that is associated with the statement when the statement is a CALL statement. 1403 class A

Input SQLDA. An SQLDA structure that describes the host variables that are used on the statement. The SQLDA structure is described in the "SQLDA" on page 573.

Isolation level. The level of record locking that occurs under commitment control. Values passed follow:

**CHG* The following are locked until the end of the unit of work (transaction):

- Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements
- Rows that are updated, deleted, and inserted

Uncommitted changes in other jobs can be seen.

**CS* The following are locked until the end of the unit of work (transaction):

- Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements
- Rows that are updated, deleted, and inserted

A row that is selected but not updated is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

**ALL*	The following are locked until the end of the unit of work (transaction): • Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements • Rows that are selected, updated, deleted, and inserted Uncommitted changes in other jobs cannot be seen.

Length of DECLARE PROCEDURE. If the statement being executed is a CALL statement and if 1 was returned for the include bound statements field when the ARD program was called using the ARCN0100 format, this field is the length of the associated DECLARE PROCEDURE statement as contained in the program. If there is no associated DECLARE PROCEDURE statement, the statement is not a CALL statement or if 0 was returned for the include bound statements field, this field is set to zero.

Length of input SQLDA. The length of the SQLDA structure that describes the input host variables that are used on the statement. If zero, no input host variables were used on the statement.

Length of procedure name. If the statement being executed is a CALL statement, this field is the length of the procedure name. Otherwise, this field is set to zero.

Length of SQL statement. If the value 1 was returned for the include bound statements field when the ARD program was called using the ARCN0100 format, this field is the length of the statement as contained in the program. Otherwise, this field is set to zero.

Offset to DECLARE PROCEDURE. If the statement being executed is a CALL statement and if 1 was returned for the include bound statements field when the ARD program was called using the ARCN0100 format, this field is the offset of the associated DECLARE PROCEDURE statement as contained in the program. If there is no associated DECLARE PROCEDURE statement, the statement is not a CALL statement, or if 0 was returned for the include bound statements field, this field is set to zero.

Offset to input SQLDA. The offset from the start of the input format structure to the SQLDA structure that describes the input host variables that are used on the statement. If zero, no input host variables were used on the statement.

Offset to procedure name. If the statement being executed is a CALL statement, this field is the offset from the start of the input format structure to the procedure name as contained in the CALL statement. Otherwise, this field is set to zero.

Offset to SQL statement. If 1 was returned for the include bound statements field when the ARD program was called using the ARCN0100 format, this field is the offset from the start of the input format structure to the SQL statement as contained in the program. Otherwise, this field is set to zero.

Output format buffer size. The amount of storage allocated for the output format that is returned by the ARD program. The length of the output format must be less than or equal to this value. It must also conform to the description of the output format associated with this input format. See the description of the output format parameter (page 542) on for a description of the output format associated with this input format.

Package collection. The collection for the package that the statement is associated with. A collection is a name that provides a logical grouping for SQL objects.

Package consistency token. The consistency token for the package that the statement is associated with. Refer to “Consistency Token” on page 570 for a description of this field.

Package name. The name of the package that the statement is associated with.

Procedure name. If the statement being executed is a CALL statement, this field contains the procedure name as specified in the CALL statement.

RDB name. The name of the relational database that the request was directed to.

Reserved. An ignored field.

Section number. The section number the statement is associated with. Refer to "Section Number" on page 571 for more information on section numbers.

SQL statement. The statement as contained in the program.

String delimiter. The statement string delimiter for the SQL statements. Values passed follow:

'	The character used as the string delimiter is the apostrophe (').
"	The character used as the string delimiter is the quotation mark (").

Time format. The format that is used when the exit program accesses time result columns. Values passed follow (where *h*=hour, *m*=minute, and *s*=second):

**USA*	The United States time format *hh:mm xx* is used, where *xx* is A.M. or P.M.
**ISO*	The International Organization for Standardization (ISO) time format *hh.mm.ss*.
**EUR*	The European time format *hh.mm.ss*.
**JIS*	The Japanese Industrial Standard time format *hh:mm:ss*.

Format ARXP0100 (Execute Prepared Statement)

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Output format buffer size
4	4	BINARY(4)	Activation group number
8	8	CHAR(18)	RDB name
26	1A	CHAR(18)	Package collection
44	2C	CHAR(18)	Package name
62	3E	CHAR(8)	Package consistency token
70	46	CHAR(2)	Reserved
72	48	BINARY(4)	Section number
76	4C	BINARY(4)	CCSID
80	50	CHAR(1)	String delimiter
81	51	CHAR(1)	Decimal delimiter
82	52	CHAR(10)	Date format
92	5C	CHAR(10)	Time format
102	66	CHAR(10)	Isolation level
112	70	BINARY(4)	Offset to SQLDA
116	74	BINARY(4)	Length of SQLDA
120	78	BINARY(4)	Offset to procedure name
124	7C	BINARY(4)	Length of procedure name

Offset		Type	Field
Dec	Hex		
128	80	CHAR(18)	Statement name
		CHAR(*)	SQLDA
		CHAR(*)	Procedure name

Field Descriptions for Format ARXP0100

Activation group number. The activation group number of the program that is performing the request. See "Activation Group" on page 570 for a description of what an activation group is.

CCSID. The CCSID of statement name.

Date format. The format that is used when the exit program accesses date result columns. Values passed follow (where *m*=month, *d*=day, and *y*=year):

**USA*	The United States date format *mm/dd/yyyy*.
**ISO*	The International Organization for Standardization (ISO) date format *yyyy-mm-dd*.
**EUR*	The European date format *dd.mm.yyyy*.
**JIS*	The Japanese Industrial Standard date format *yyyy-mm-dd*.

Decimal delimiter. The statement decimal delimiter for the SQL statements. Values passed follow:

.	The value used as the decimal point in numeric literals is a period.
,	The value used as the decimal point in numeric literals is a comma.

Isolation level. The level of record locking that occurs under commitment control. Values passed follow:

**CHG*
The following are locked until the end of the unit of work (transaction):

- Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements
- Rows that are updated, deleted, and inserted

Uncommitted changes in other jobs can be seen.

**CS*
The following are locked until the end of the unit of work (transaction):

- Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements
- Rows that are updated, deleted, and inserted

A row that is selected but not updated is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

**ALL*
The following are locked until the end of the unit of work (transaction):

- Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements
- Rows that are selected, updated, deleted, and inserted

Uncommitted changes in other jobs cannot be seen.

Length of procedure name. If the statement being executed is a CALL statement, this field is the length of the procedure name. This field will always be set to 0; it is reserved for future use.

Length of SQLDA. The length of the SQLDA structure that describes the host variables that are used on the statement. If zero, no host variables were used on the statement.

Offset to procedure name. If the statement being executed is a CALL statement, this field is the offset from the start of the input format structure to the procedure name as contained in the CALL statement. This field will always be set to 0; it is reserved for future use.

Offset to SQLDA. The offset from the start of the input format structure to the SQLDA structure that describes the host variables that are used on the statement. If zero, no host variables were used on the statement.

Output format buffer size. The amount of storage allocated for the output format that is returned by theARD program. The length of the output format must be less than or equal to this value. It must also conform to the description of the output format associated with this input format. See the description of the output format parameter (page 542) on for a description of the output format associated with this input format.

Package collection. The collection for the package that the statement is associated with. A collection is a name that provides a logical grouping for SQL objects.

Package consistency token. The consistency token for the package that the statement is associated with. Refer to “Consistency Token” on page 570 for a description of this field.

Package name. The name of the package that the statement is associated with.

Procedure name. If the statement being executed is a CALL statement, this field contains the procedure name as specified in the CALL statement. This field will not be passed; it is reserved for future use.

RDB name. The name of the relational database that the request was directed to.

Reserved. An ignored field.

Section number. The section number that the statement is associated with. Refer to “Section Number” on page 571 for more information on section numbers.

SQLDA. An SQLDA structure that describes the host variables that are used on the statement. The SQLDA structure is described in the “SQLDA” on page 573.

Statement name. The SQL statement name that is specified on the EXECUTE statement.

String delimiter. The statement string delimiter for the SQL statements. Values passed follow:

′	The character used as the string delimiter is the apostrophe (′).
″	The character used as the string delimiter is the quotation mark (“).

Time format. The format that is used when the exit program accesses time result columns. Values passed follow (where *h*=hour, *m*=minute, and *s*=second):

**USA*	The United States time format *hh:mm xx* is used, where *xx* is A.M. or P.M.
**ISO*	The International Organization for Standardization (ISO) time format *hh.mm.ss*.
**EUR*	The European time format *hh.mm.ss*.
**JIS*	The Japanese Industrial Standard time format *hh:mm:ss*.

Format ARXI0100 (Execute Immediate Statement Format)

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Output format buffer size
4	4	BINARY(4)	Activation group number
8	8	CHAR(18)	RDB name
26	1A	CHAR(18)	Package collection
44	2C	CHAR(18)	Package name
62	3E	CHAR(8)	Package consistency token
70	46	CHAR(2)	Reserved
72	48	BINARY(4)	Section number
76	4C	BINARY(4)	CCSID
80	50	CHAR(1)	String delimiter
81	51	CHAR(1)	Decimal delimiter
82	52	CHAR(10)	Date format
92	5C	CHAR(10)	Time format
102	66	CHAR(10)	Isolation level
112	70	BINARY(4)	Offset to SQL statement
116	74	BINARY(4)	Length of SQL statement
		CHAR(*)	SQL statement

Field Descriptions for Format ARXI0100

Activation group number. The activation group number of the program that is performing the request. See "Activation Group" on page 570 for a description of what an activation group is.

CCSID. The CCSID of the statement text.

Date format. The format that is used when the exit program accesses date result columns. Values passed follow (where *m*=month, *d*=day, and *y*=year):

**USA*	The United States date format *mm/dd/yyyy*.
**ISO*	The International Organization for Standardization (ISO) date format *yyyy-mm-dd*.
**EUR*	The European date format *dd.mm.yyyy*.
**JIS*	The Japanese Industrial Standard date format *yyyy-mm-dd*.

Decimal delimiter. The statement decimal delimiter for the SQL statements. Values passed follow:

.	The value used as the decimal point in numeric literals is a period.
,	The value used as the decimal point in numeric literals is a comma.

Isolation level. The level of record locking that occurs under commitment control. Values passed follow:

**CHG*	The following are locked until the end of the unit of work (transaction): • Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements • Rows that are updated, deleted, and inserted Uncommitted changes in other jobs can be seen.
**CS*	The following are locked until the end of the unit of work (transaction): • Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements • Rows that are updated, deleted, and inserted A row that is selected but not updated is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.
**ALL*	The following are locked until the end of the unit of work (transaction): • referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements • Rows that are selected, updated, deleted, and inserted Uncommitted changes in other jobs cannot be seen.

Length of SQL statement. The length of the SQL statement to execute.

Offset to SQL statement. The offset from the start of the input format structure to the SQL statement to execute.

Output format buffer size. The amount of storage allocated for the output format that is returned by the ARD program. The length of the output format must be less than or equal to this value. It must also conform to the description of the output format associated with this input format. See the description of the output format parameter (page 542) on for a description of the output format associated with this input format.

Package collection. The collection for the package that the statement is associated with. A collection is a name that provides a logical grouping for SQL objects.

Package consistency token. The consistency token for the package that the statement is associated with. Refer to "Consistency Token" on page 570 for a description of this field.

Package name. The name of the package that the statement is associated with.

RDB name. The name of the relational database that the request was directed to.

Reserved. An ignored field.

Section number. The section number that the statement is associated with. Refer to "Section Number" on page 571 for more information on section numbers.

SQL statement. The SQL statement to execute.

String delimiter. The statement string delimiter for the SQL statements. Values passed follow:

′	The character used as the string delimiter is the apostrophe (′).
″	The character used as the string delimiter is the quotation mark (“).

Time format. The format that is used when the exit program accesses time result columns. Values passed follow (where *h*=hour, *m*=minute, and *s*=second):

**USA*	The United States time format *hh:mm xx* is used, where *xx* is A.M. or P.M.
**ISO*	The International Organization for Standardization (ISO) time format *hh.mm.ss*.
**EUR*	The European time format *hh.mm.ss*.
**JIS*	The Japanese Industrial Standard time format *hh:mm:ss*.

Format AROC0100 (Open Cursor Format)

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Output format buffer size
4	4	BINARY(4)	Activation group number
8	8	CHAR(18)	RDB name
26	1A	CHAR(18)	Package collection
44	2C	CHAR(18)	Package name
62	3E	CHAR(8)	Package consistency token
70	46	CHAR(2)	Reserved
72	48	BINARY(4)	Section number
76	4C	BINARY(4)	CCSID
80	50	CHAR(1)	String delimiter
81	51	CHAR(1)	Decimal delimiter
82	52	CHAR(10)	Date format
92	5C	CHAR(10)	Time format
102	66	CHAR(10)	Isolation level
112	70	CHAR(18)	Default collection
130	82	CHAR(1)	Blocking allowed
131	83	CHAR(1)	Reserved
132	84	BINARY(4)	Offset to SQLDA
136	88	BINARY(4)	Length of SQLDA
140	8C	BINARY(4)	Offset to DECLARE CURSOR
144	90	BINARY(4)	Length of DECLARE CURSOR
148	94	CHAR(18)	Cursor name
		CHAR(*)	SQLDA
		CHAR(*)	DECLARE CURSOR statement

Field Descriptions for Format AROC0100

Activation group number. The activation group number of the program that is performing the request. See "Activation Group" on page 570 for a description of what an activation group is.

Blocking allowed. Whether blocking should be performed for the cursor. Values passed follow:

0	Blocking is not allowed.
1	Blocking is allowed.

CCSID. The CCSID of the statement text and cursor name.

Cursor name. The cursor name that is specified on the OPEN statement.

Date format. The format that is used when the exit program accesses date result columns. Values passed follow (where *m*=month, *d*=day, and *y*=year):

**USA*	The United States date format *mm/dd/yyyy*.
**ISO*	The International Organization for Standardization (ISO) date format *yyyy-mm-dd*.
**EUR*	The European date format *dd.mm.yyyy*.
**JIS*	The Japanese Industrial Standard date format *yyyy-mm-dd*.

DECLARE CURSOR statement. The DECLARE CURSOR statement as contained in the program that is associated with the OPEN statement.

Decimal delimiter. The statement decimal delimiter for the SQL statements. Values passed follow:

.	The value used as the decimal point in numeric literals is a period.
,	The value used as the decimal point in numeric literals is a comma.

Isolation level. The level of record locking that occurs under commitment control. Values passed follow:

**CHG* The following are locked until the end of the unit of work (transaction):

- Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements
- Rows that are updated, deleted, and inserted

Uncommitted changes in other jobs can be seen.

**CS* The following are locked until the end of the unit of work (transaction):

- Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements
- Rows that are updated, deleted, and inserted

A row that is selected but not updated is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

**ALL* The following are locked until the end of the unit of work (transaction):

- Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements
- Rows that are selected, updated, deleted, and inserted

Uncommitted changes in other jobs cannot be seen.

Length of DECLARE CURSOR. If 1 was returned for the include bound statements field when the ARD program was called using the ARCN0100 format, this field is the length of the associated DECLARE CURSOR statement as contained in the program. Otherwise, this field is set to zero.

Length of SQLDA. The length of the SQLDA structure that describes the host variables that are used on the statement. If zero, no host variables were used on the statement.

Offset to DECLARE CURSOR. If 1 was returned for the include bound statements field when the ARD program was called using the ARCN0100 format, this field is the offset of the associated DECLARE CURSOR statement as contained in the program. Otherwise, this field is set to zero.

Offset to SQLDA. The offset from the start of the input format structure to the SQLDA structure that describes the host variables that are used on the statement. If zero, no host variables were used on the statement.

Output format buffer size. The amount of storage allocated for the output format that is returned by the ARD program. The length of the output format must be less than or equal to this value. It must also conform to the description of the output format associated with this input format. See the description of the output format parameter (page 542) on for a description of the output format associated with this input format.

Package collection. The collection for the package that the statement is associated with. A collection is a name that provides a logical grouping for SQL objects.

Package consistency token. The consistency token for the package that the statement is associated with. Refer to "Consistency Token" on page 570 for a description of this field.

Package name. The name of the package that the statement is associated with.

RDB name. The name of the relational database that the request was directed to.

Reserved. An ignored field.

Section number. The section number that the statement is associated with. Refer to "Section Number" on page 571 for more information on section numbers.

SQLDA. An SQLDA structure that describes the host variables that are used on the statement. The SQLDA structure is described in the "SQLDA" on page 573.

String delimiter. The statement string delimiter for the SQL statements. Values passed follow:

′	The character used as the string delimiter is the apostrophe (′).
″	The character used as the string delimiter is the quotation mark (“).

Time format. The format that is used when the exit program accesses time result columns. Values passed follow (where *h*=hour, *m*=minute, and *s*=second):

**USA*	The United States time format *hh:mm xx* is used, where *xx* is A.M. or P.M.
**ISO*	The International Organization for Standardization (ISO) time format *hh.mm.ss*.
**EUR*	The European time format *hh.mm.ss*.
**JIS*	The Japanese Industrial Standard time format *hh:mm:ss*.

Format ARFC0100 (Fetch from a Cursor Format)

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Output format buffer size
4	4	BINARY(4)	Activation group number
8	8	CHAR(18)	RDB name
26	1A	CHAR(18)	Package collection
44	2C	CHAR(18)	Package name
62	3E	CHAR(8)	Package consistency token

Offset			
Dec	**Hex**	**Type**	**Field**
70	46	CHAR(2)	Reserved
72	48	BINARY(4)	Section number
76	4C	BINARY(4)	CCSID
80	50	CHAR(1)	String delimiter
81	51	CHAR(1)	Decimal delimiter
82	52	CHAR(10)	Date format
92	5C	CHAR(10)	Time format
102	66	CHAR(10)	Isolation level
112	70	CHAR(18)	Cursor name

Field Descriptions for Format ARFC0100

Activation group number. The activation group number of the program that is performing the request. See "Activation Group" on page 570 for a description of what an activation group is.

CCSID. The CCSID of the cursor name.

Cursor name. The cursor name that is specified on the FETCH statement.

Date format. The format that is used when the exit program accesses date result columns. Values passed follow (where *m*=month, *d*=day, and *y*=year):

**USA*	The United States date format *mm/dd/yyyy*.
**ISO*	The International Organization for Standardization (ISO) date format *yyyy-mm-dd*.
**EUR*	The European date format *dd.mm.yyyy*.
**JIS*	The Japanese Industrial Standard date format *yyyy-mm-dd*.

Decimal delimiter. The statement decimal delimiter for the SQL statements. Values passed follow:

.	The value used as the decimal point in numeric literals is a period.
,	The value used as the decimal point in numeric literals is a comma.

Isolation level. The level of record locking that occurs under commitment control. Values passed follow:

**CHG* The following are locked until the end of the unit of work (transaction):

- Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements
- Rows that are updated, deleted, and inserted

Uncommitted changes in other jobs can be seen.

**CS* The following are locked until the end of the unit of work (transaction):

- Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements
- Rows that are updated, deleted, and inserted

A row that is selected but not updated is locked until the next row is selected. Uncommitted changes in other jobs cannot be seen.

**ALL*	The following are locked until the end of the unit of work (transaction): • Objects that are referred to in SQL ALTER, COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements • Rows that are selected, updated, deleted, and inserted Uncommitted changes in other jobs cannot be seen.

Output format buffer size. The amount of storage allocated for the output format that is returned by the ARD program. The length of the output format must be less than or equal to this value. It must also conform to the description of the output format associated with this input format. See the description of the output format parameter (page 542) on for a description of the output format associated with this input format.

Package collection. The collection for the package that the statement is associated with. A collection is a name that provides a logical grouping for SQL objects.

Package consistency token. The consistency token for the package that the statement is associated with. Refer to “Consistency Token” on page 570 for a description of this field.

Package name. The name of the package that the statement is associated with.

RDB name. The name of the relational database that the request was directed to.

Reserved. An ignored field.

Section number. The section number that the statement is associated with. Refer to “Section Number” on page 571 for more information on section numbers.

String delimiter. The statement string delimiter for the SQL statements. Values passed follow:

′	The character used as the string delimiter is the apostrophe (′).
″	The character used as the string delimiter is the quotation mark (“).

Time format. The format that is used when the exit program accesses time result columns. Values passed follow (where *h*=hour, *m*=minute, and *s*=second):

**USA*	The United States time format *hh:mm xx* is used, where *xx* is A.M. or P.M.
**ISO*	The International Organization for Standardization (ISO) time format *hh.mm.ss*.
**EUR*	The European time format *hh.mm.ss*.
**JIS*	The Japanese Industrial Standard time format *hh:mm:ss*.

Format ARCC0100 (Close a Cursor Format)

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Output format buffer size
4	4	BINARY(4)	Activation group number
8	8	CHAR(18)	RDB name
26	1A	CHAR(18)	Package collection
44	2C	CHAR(18)	Package name

Offset		Type	Field
Dec	Hex		
62	3E	CHAR(8)	Package consistency token
70	46	CHAR(2)	Reserved
72	48	BINARY(4)	Section number
76	4C	BINARY(4)	CCSID
80	50	CHAR(18)	Cursor name

Field Descriptions for Format ARCC0100

Activation group number. The activation group number of the program that is performing the request. See "Activation Group" on page 570 for a description of what an activation group is.

CCSID. The CCSID of the cursor name.

Cursor name. The cursor name specified on the CLOSE statement.

Output format buffer size. The amount of storage allocated for the output format that is returned by the ARD program. The length of the output format must be less than or equal to this value. It must also conform to the description of the output format associated with this input format. See the description of the output format parameter (page 542) on for a description of the output format associated with this input format.

Package collection. The collection for the package that the statement is associated with. A collection is a name that provides a logical grouping for SQL objects.

Package consistency token. The consistency token for the package that the statement is associated with. Refer to "Consistency Token" on page 570 for a description of this field.

Package name. The name of the package that the statement is associated with.

RDB name. The name of the relational database that the request was directed to.

Reserved. An ignored field.

Section number. The section number that the statement is associated with. Refer to "Section Number" on page 571 for more information on section numbers.

Format ARDS0100 (Describe a Statement Format)

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Output format buffer size
4	4	BINARY(4)	Activation group number
8	8	CHAR(18)	RDB name
26	1A	CHAR(18)	Package collection
44	2C	CHAR(18)	Package name
62	3E	CHAR(8)	Package consistency token
70	46	CHAR(2)	Reserved

Offset		Type	Field
Dec	Hex		
72	48	BINARY(4)	Section number
76	4C	BINARY(4)	CCSID
80	50	CHAR(18)	Statement name

Field Descriptions for Format ARDS0100

Activation group number. The activation group number of the program that is performing the request. See "Activation Group" on page 570 for a description of what an activation group is.

CCSID. The CCSID of the statement name.

Output format buffer size. The amount of storage allocated for the output format that is returned by the ARD program. The length of the output format must be less than or equal to this value. It must also conform to the description of the output format associated with this input format. See the description of the output format parameter (page 542) on for a description of the output format associated with this input format.

Package collection. The collection for the package that the statement is associated with. A collection is a name that provides a logical grouping for SQL objects.

Package consistency token. The consistency token for the package that the statement is associated with. Refer to "Consistency Token" on page 570 for a description of this field.

Package name. The name of the package that the statement is associated with.

RDB name. The name of the relational database that the request was directed to.

Reserved. An ignored field.

Section number. The section number that the statement is associated with. Refer to "Section Number" on page 571 for more information on section numbers.

Statement name. The statement name that is specified on the DESCRIBE statement.

Format ARDT0100 (Describe Object Format)

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Output format buffer size
4	4	BINARY(4)	Activation group number
8	8	CHAR(18)	RDB name
26	1A	CHAR(2)	Reserved
28	1C	BINARY(4)	CCSID
32	20	BINARY(4)	Offset to object name
36	24	BINARY(4)	Length of object name
		CHAR(*)	Object name

Field Descriptions for Format ARDT0100

Activation group number. The activation group number of the program that is performing the request. See “Activation Group” on page 570 for a description of what an activation group is.

CCSID. The CCSID of the object name.

Length of object name. The length of the name of the SQL object to describe.

Object name. The name of the SQL object to be described. This may be either a qualified or unqualified table name. If it is qualified, it will be in the SQL naming format, that is, a collection name followed by a period and an SQL identifier.

Offset to object name. The offset from the start of the input format structure to the name of the SQL object to describe.

Output format buffer size. The amount of storage allocated for the output format that is returned by the ARD program. The length of the output format must be less than or equal to this value. It must also conform to the description of the output format associated with this input format. See the description of the output format parameter (page 542) on for a description of the output format associated with this input format.

RDB name. The name of the relational database that the request was directed to.

Reserved. An ignored field.

Output Format Structures

In the following structures, the CCSID of the character fields is the job CCSID unless a specific CCSID field is included in the format for the field.

Output Connect Format

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(3)	Server product
3	3	CHAR(2)	Server version
5	5	CHAR(2)	Server release
7	7	CHAR(1)	Server level
8	8	CHAR(10)	User ID
18	12	CHAR(1)	Include bound statements
19	13	CHAR(1)	Protected conversation

Field Descriptions for Output Connect Format

Include bound statements. Whether statements that were sent at package-creation time should be included in run-time formats ARXD0100, ARXB0100, and AROC0100. Valid values follow:

0	Do not include bound statements.
1	Include bound statements.

Protected conversation. Whether a protected conversation is used for the connection. If the connection uses a protected conversation, the system rejects attempts by the application to run the DISCONNECT SQL statement. Connections that use protected conversations are only ended during commit and rollback processing. See the description of the disconnect type field (page Disconnect type (page 544)) for format ARDI0100 for more information. Valid values follow:

0	Conversation is not protected.
1	Conversation is protected.

Server level. An identifier for the level of the database server that is accessed by the ARD program. This value must be a character representation of a hexadecimal value. That is, it may only consist of the characters 0-9 and A-F.

Server product. An identifier for the database server that is accessed by the ARD program. The system does no verification of the value of this field.

Server release. An identifier for the release of the database server that is accessed by the ARD program. This value must be a character representation of a hexadecimal value. That is, it may only consist of the characters 0-9 and A-F.

Server version. An identifier for the version of the database server that is accessed by the ARD program. This value must be a character representation of a hexadecimal value. That is, it may only consist of the characters 0-9 and A-F. The system does no verification of the value of this field.

User ID. The user ID that is used at the server.

Output Execute Format

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(1)	Update performed
1	1	CHAR(3)	Reserved
4	4	BINARY(4)	Offset to SQLDA
8	8	BINARY(4)	Offset to result set
		CHAR(*)	SQLDA
		CHAR(*)	Result set

Field Descriptions for Output Execute Format

Offset to result set. The offset from the start of the output format to the result set. If the SQLCA indicates an error occurred, this field must be set to 0.

Offset to SQLDA. An offset from the start of the output format to the SQLDA structure that describes the columns for the results of the statement. This field can only have a value of 0 or a multiple of 16. If the SQLCA indicates an error occurred, this field must be set to 0. If the SQLCA does not indicate an error, this field cannot be 0.

Reserved. An ignored field.

Result set. The result for the SQL statement. Columns are contiguous with null indicators (if appropriate) that precede the column data. Refer to “Query (Fetch) Data Format” on page 572 for more information. If the offset to result set field is 0, this field must not be included in the output format.

Note: This null indicator is not the same as the NULL in the C language.

SQLDA. An SQLDA structure that describes the columns for the results of the statement. The SQLDA structure is described in "SQLDA" on page 573. If the offset to SQLDA field is 0, this field must not be included in the output format.

Update performed. Whether the statement resulted in an update. An update is any operation that results in a change to an object such that the object is under commitment control. Valid values follow:

0	The operation did not result in an update.
1	An update occurred from the operation.

Output Open Cursor Format

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(1)	Block data
1	1	CHAR(1)	Cursor held
1	1	CHAR(2)	Reserved
4	4	BINARY(4)	Offset to SQLDA
		CHAR(*)	SQLDA

Field Descriptions for Output Open Cursor Format

Block data. Whether the ARD program will block the data. Valid values follow:

0	A single row of data will be returned.
1	Data will be returned in blocks.

Cursor held. Whether the cursor is held open after commits. Valid values follow:

0	Cursor is closed after commits.
1	Cursor is held open after commits.

Offset to SQLDA. The offset from the start of the output format to the SQLDA structure that describes the columns for the results of the statement. This field can only have a value of 0 or a multiple of 16. If the SQLCA indicates that an error occurred, this field must be set to 0. If the SQLCA does not indicate an error, this field cannot be 0.

Reserved. An ignored field.

SQLDA. An SQLDA structure that describes the columns for the results of the statement. The SQLDA structure is described in "SQLDA" on page 573. If the offset to SQLDA field is 0, this field must not be included in the output format.

Output Fetch Cursor Format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Offset to result set
4	4	CHAR(1)	Cursor closed
		CHAR(*)	Result set

Field Descriptions for Output Fetch Cursor Format

Cursor closed. Whether the cursor is closed. Valid values follow:

0	The cursor is open.
1	The cursor is closed.

Offset to result set. The offset from the start of the output format to the result set. If no data is returned, this field should be set to 0.

Result set. The result for the SQL statement. Columns are contiguous with null indicators (if appropriate) that precede the column data. Refer to "Query (Fetch) Data Format" on page 572 for more information. If the offset to result set field is 0, this field must not be included in the output format.

Note: This null indicator is not the same as the NULL in the C language.

Activation Group

An activation group provides the following:

- Run-time data structures to support the running of programs
- Addressing protection
- A logical boundary for message creation
- A logical boundary for application cleanup processing

Connections are scoped to the activation group. Therefore, the activation group mark and the RDB name together are used to uniquely identify the connection. It is not possible to have more than one connection active with the same RDB name in the same activation group at a point in time. However, it is possible to have multiple connections with different RDB names in the same activation group and to have multiple connections with the same RDB name in different activation groups.

Consistency Token

The system associates a consistency token with every program. If a program is compiled again, a new consistency token is created. When a user or application creates a package with the CRTSQLxxx commands or the CRTSQLPKG command and an RDB that is associated with an ARD program is specified on the command, the package consistency token from the program along with a package name and package collection is passed to the ARD program. In addition, at program run time, the ARD program will be passed the package name, package collection, and package consistency token that are currently associated with the program.

The ARD program can use this information passed to it during run time to verify that information passed to it during package creation is correct for the instance of the program being run. If a package does not exist for the given package consistency token, package name, and package collection, the exit program should return messages SQLCODE (-805) and SQLSTATE (51002).

Section Number

When a user or application creates a package with the CRTSQLxxx commands or the CRTSQLPKG command and an RDB that is associated with an ARD program is specified on the command, statements contained in the program are passed to the ARD program. A section number is associated with the statements. A section number is a signed binary number ranging from 1 to 32767. Section numbers may not necessarily be consecutive.

Related statements share the same section numbers. Therefore, a cursor declared for a statement and each statement that references the declared statement or cursor (FETCH, EXECUTE, OPEN, CLOSE, PREPARE) have the same section number. However, each uniquely declared statement or cursor has a different section number.

The system assigns a unique section number to the following statements and any other statements that it does not understand:

- ALTER
- COMMENT ON
- CREATE
- DELETE
- DROP
- EXPLAIN
- GRANT
- INSERT
- LABEL ON
- LOCK
- REVOKE
- SELECT (embedded)
- SET
- UPDATE
- EXECUTE IMMEDIATE
- CALL
- DECLARE PROCEDURE

The following statements are not passed to the ARD program during the package creation process. Also, local statements that are understood by the precompiler but do not result in calls to the ARD program at run time are not passed.

- INCLUDE
- WHENEVER
- PREPARE
- EXECUTE
- EXECUTE IMMEDIATE
- DESCRIBE
- OPEN
- FETCH
- CLOSE
- COMMIT
- CONNECT
- ROLLBACK
- RELEASE

- SET CONNECTION
- DISCONNECT
- BEGIN DECLARE SECTION
- END DECLARE SECTION

Query (Fetch) Data Format

Query data is returned as a continuous collection of columns. Multiple rows of data may also be returned for format ARFC0100 when the ARD program returned 1 for the block data field on format AROC0100. When multiple rows of data are returned, the rows are also contiguous.

If a column is a null-capable or a derived field (for example COL1/COL2), the column data is preceded with a 1-byte null indicator. The length of data returned for variable-length data types should be based on the length in the length indicator. It should not be padded to the length of the field. Representation of all data types is assumed to be in the format used by OS/400.

The null indicator is a 1-byte signed binary integer. If the null indicator is negative (between X'80' and X'FF', inclusive), no column data should follow the indicator. For data conversion errors, -2 (X'FE') should be used for the null indicator.

If a data conversion error occurs and the column is non-null-capable, an error sqlcode should be returned in the SQLCA. When an error (negative SQLCODE) is indicated in the SQLCA and the ARD program is not returning multiple rows, no data should be returned. When multiple rows of data are being returned for a query and the ARD program returns an error in the SQLCA, the row that the error applies to should not be included in the block and the row previous to the row in error should be the last row returned.

When a warning (positive SQLCODE) is indicated in the SQLCA and the ARD program is not returning multiple rows, the row should be returned. When multiple rows of data are being returned for a query and an ARD program returns a warning in the SQLCA, the row that the warning applies to should be the last row in the block.

If the ARD program has not indicated that the cursor is closed in the cursor closed field of the output format for ARFC0100, the system will call the ARD program to get additional rows when the application requests a row after the row that the warning or error applies to. If the ARD program indicated the cursor was closed, the system will not call the ARD program for that cursor again until the application performs another SQL OPEN.

The following illustration shows an example of two rows of data being returned for a FETCH. The two rows each consist of a null-capable smallint COL1, a non-null-capable CHAR(3) COL2, a null-capable smallint COL3, and a non-null-capable VARCHAR(20) COL4.

```
Hex Representation           Description
00                    Row 1 - Null byte for COL1 - not null
0001                  Row 1 - COL1 (smallint) value = 1
D1E6E3                Row 1 - COL2 (CHAR(3)) value =  JWT
FF                    Row 1 - Null byte for COL3 - null
0007D1C5C6C6D9C5E8    Row 1 - COL4 Length = 7 value = JEFFREY
FF                    Row 2 - Null byte for COL1 - null
D1D4C2                Row 2 - COL2 (CHAR(3)) value =  JMB
00                    Row 2 - Null byte for COL3 - not null
0002                  Row 2 - COL3 (smallint) value = 2
0004D1D6C8D5          Row 1 - COL4 Length = 4 value = JOHN
```

In response to format ARXD0100 when data is returned for a CALL statement, a null indicator must precede each field regardless of whether the field is null-capable or not. For any parameters declared as input-only on the DECLARE PROCEDURE statement, the null indicator must be set to X'80'.

The following illustration shows an example of the data returned for a CALL where the first parameter was an input-only parameter and the second was an output smallint.

```
Hex Representation           Description
80                   Parm 1 - Null byte - input only
00                   Parm 2 - Null byte - not null
0001                 Parm 2 - Value = 1
```

SQLDA

An SQLDA is a set of variables that describe either host variables or column attributes. Included in this topic are the SQLDA structure, the relevant settings for those fields that the ARD program returns to the operating system, and the relevant fields that are passed to the ARD program.

For more information about SQLDA, see SQL Descriptor Area (SQLDA) in the DB2 UDB for iSeries SQL Reference topic.

The SQLDA has the following fields (shown in C-language format):

```
struct sqlda
{
    unsigned char  sqldaid[8];
    long           sqldabc;
    short          sqln;
    short          sqld;
    struct sqlvar
    {
        short         sqltype;
        short         sqllen;
        unsigned char sqlres[12];
        unsigned char *sqldata;
        short         *sqlind;
        struct sqlname
        {
            short        length;
            unsigned char data[30];
        } sqlname;
    } sqlvar[1];
};
```

In response to the ARPD0100, ARDS0100, and ARDT0100 formats, the following fields must be set on the return from the ARD program:

sqldaid	An eye-catcher for diagnostic purposes. This must be set to 'SQLDA'.
sqldabc	The length of the SQLDA. Its value is calculated as '2*sqld*sizeof(sqlvar) + 16'.
sqld	A number equal to the number of columns described. The actual number of sqlvar occurrences returned should be twice this number. If the statement being described is not a SELECT statement, this field is set to 0.

sqlvar	A structure that contains two entries for each column in the result table. If *n*is the number of columns being described, the first *n* sqlvar entries contain the following:

sqltype	The sqltype of the column in the select list of the result table. See the DB2 UDB for iSeries SQL Reference topic for a complete list of field data types and their corresponding sqltype value.
sqllen	The length attribute of the column.
sqldata	For character data, the CCSID of the field in the following format: bytes 1 and 2 are set to X'00' and bytes 3 and 4 are set to the CCSID value.
sqlind	Reserved. This must be set to X'00'.
sqlname	The unqualified name of the column.

sqlname.length
The length of the unqualified name of the column.

sqlname.data
The unqualified column name. The CCSID of the value is the CCSID of the job.

And the second *n* entries contain the following:

sqltype	Reserved. This must be set to X'00'.
sqllen	Reserved. This must be set to X'00'.
sqldata	Reserved. This must be set to X'00'.
sqlind	Reserved. This must be set to X'00'.
sqlname	The label of the column.

sqlname.length
The length of the column label.

sqlname.data
The column label. The CCSID of the value is the CCSID of the job.

In response to formats AROC0100 and ARXD0100, the following fields must be set on the return from the ARD program:

sqldaid	An eye-catcher for diagnostic purposes. This must be set to 'SQLDA'.
sqldabc	The length of the SQLDA. Its value is calculated as 'sqld*sizeof(sqlvar) + 16'.
sqld	The number of columns in the result table.
sqlvar	A structure that contains an entry for each column in the result table. The entries contain the following fields:

sqltype	The sqltype of the column in the select list of the result table. See the DB2 UDB for iSeries SQL Reference topic for a complete list of field data types and their corresponding sqltype value.
sqllen	The length attribute of the column.
sqldata	For character data, the CCSID of the field in the following format: bytes 1 and 2 are set to X'00' and bytes 3 and 4 are set to the CCSID value.
sqlind	Reserved. This must be set to X'00'.
sqlname	The unqualified name of the column.

sqlname.length
The length of the unqualified name of the column.

sqlname.data
The unqualified column name. The CCSID of the value is the CCSID of the job.

On input to the ARD program on formats ARBS0100, ARXD0100, ARXB0100, ARXP0100, and AROC0100, the ARD program must interpret the SQLDA because it describes host variables to be used with the statement. The relevant fields are:

sqld
The number of host variables for the statement.

sqlvar
A structure that contains one entry for each host variable. The entries contain the following:

sqltype The sqltype of the host variable. See the DB2 UDB for iSeries SQL Reference topic for a complete list of field data types and their corresponding sqltype value. An odd value for sqltype indicates that this sqlvar entry contains a pointer to an indicator variable that is addressed by sqlind.

sqllen The length attribute of the host variable.

sqldata A pointer to the host variable data. This field is set to NULL for format ARBS0100.

sqlind A pointer to an indicator variable. The indicator variable is a 2-byte binary value that signifies a NULL value when sqlind is set to a negative value. This field is only relevant if the sqltype field is odd. This field is set to NULL for format ARBS0100.

sqlname The coded character set identifier (CCSID) for character host variables.

sqlname.data
The CCSID of the field in the following format: bytes 1 and 2 are set to X’00’ and bytes 3 and 4 are set to the CCSID value.

sqlname.length
This field is set to 8.

Commit APIs

To process commit and rollback requests, providers of ARD programs must register a commitment control resource. Refer to “Commit APIs” for more information on the commitment control APIs. The following section assumes an understanding of the commit APIs.

The Add Commitment Resource (QTNADDCR) API should be called to add a commitment resource to a commitment definition. After the resource is added, the exit program specified on the Add Commitment Resource API is called during commitment control operations for the commitment definition. When registering commitment resources for use with ARD programs, the commitment resource should be added after the first successful operation after the ARCN0100 format call. It is best not to perform this operation as part of the ARCN0100 format call since once a commitment control API resource is registered, the commitment definition is no longer at a logical unit-of-work boundary. A CONNECT operation does not normally change the logical unit-of-work boundary of a commitment definition. If a commitment resource is registered during the ARCN0100 format call, the create SQL package (CRTSQLPKG) function fails.

The Remove Commitment Resource (QTNRMVCR) API should be called to remove a resource from a commitment definition. This API should be called after processing the ARDI0100 format call. It cannot be called during the ARDI0100 format call if the ARDI0100 format indicates that the disconnection is occurring as part of a commit or rollback. A disconnect type of 3 indicates that the disconnect is occurring as part of a commit or rollback. In this situation, the commit resource should be removed either during the next SQL operation the ARD program processes or during the next commit or rollback operation. In the latter case, the commitment control exit program can have the resource removed by using the changes ended field in the return information format. Until this resource is removed, the Create SQL Package (CRTSQLPKG) command will fail.

Exit program introduced: V3R6

Top | “Database and File APIs,” on page 1 | APIs by category

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

```
IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY  10504-1785
U.S.A.
```

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

```
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan
```

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Trademarks

Terms and conditions for downloading and printing publications

Permissions for the use of the information you have selected for download are granted subject to the following terms and conditions and your indication of acceptance thereof.

Personal Use: You may reproduce this information for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of this information, or any portion thereof, without the express consent of IBM$^{(R)}$.

Code disclaimer information

Printed in USA