

APIs by Example: Hidden Job SQL Information Exposed by Retrieve Job Information API

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 04/28/2011 (All day)



In the [preceding installment of the APIs by Example column](#), I showed an example of how APIs at times offer access to more detailed information than corresponding system CL commands. I also demonstrated how you, as an API programmer, can add more functionality and other enhancements when you create your own API-based versions of system CL commands. The Display Job Open Files

(DSPJOBOPNF) command presented last time is today accompanied by yet another example of exploiting system APIs' access to useful information not easily obtained elsewhere.

Although the Display Job (DSPJOB) and Work with Job (WRKJOB) commands and their related display panels have been enhanced repeatedly over time in order to reflect the conceptual changes and functional enhancements provided for the IBM i OS job entity—such as activation groups, mutexes, and threads—for some reason, IBM has not yet offered much detail as far as job SQL information is concerned—in terms of the mentioned CL commands, that is. The Retrieve Job Information (QUSRJOB) API, however, has for some time been supporting a return format exposing a job's SQL-related information. And with release 6.1, this offering has been significantly enhanced with many new job SQL attributes.

The QUSRJOB API job information return format name for SQL information is JOBI0900, and this format is valid only for active jobs. For jobs waiting on a job queue or jobs that have completed, no SQL information is available. I'm using the JOBI0900 SQL information format as the foundation for the Display Job SQL Information (DSPJOBSQLI) command that I've created to accompany today's APIs by Example article. Since release 6.1's predecessor, release 5.4, also added vital information to the JOBI0900 return format, the DSPJOBSQLI command was designed to support this release as the earliest.

While developing the DSPJOBSQLI command, however, I ran into some troubles concerning the data actually returned by the QUSRJOB API for the JOBI0900 return format. For release 5.4 and, for example, prestart jobs waiting to become active, it turned out that some parts of the JOBI0900 format contained "garbage" data. I've reported the issue to IBM, and as of this writing, I'm still waiting for IBM's response. It appears, though, that the issue has disappeared on release 6.1, so for now I've coded a workaround to ensure that the DSPJOBSQLI CPP does not fail due to invalid data when run on release 5.4.

The QUSRJOB API parameter interface as such is quite simple, yet the API is capable of accessing lots of job information, all divided into currently 12 different return formats, at release 7.1:

JOBI0100	Basic performance information
JOBI0150	Additional performance information

JOBIO200	WRKACTJOB information
JOBIO300	Job queue and output queue information
JOBIO400	Job attribute information
JOBIO500	Message logging information
JOBIO600	Active job information
JOBIO700	Library list information
JOBIO750	Extended library list information
JOBIO800	Active job signal information
JOBIO900	Active job SQL information
JOBIO1000	Elapsed performance statistics

As mentioned, today's article covers the use of format JOBIO900 and the active job SQL information provided by that format. For more details on the many other formats, I suggest you follow the link at the end of this article pointing to the QUSRJOB API documentation in the IBM i Information Center. As for the QUSRJOB API, I've included the parameter list from said documentation below:

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	Qualified job name	Input	Char(26)
5	Internal job identifier	Input	Char(16)

Optional Parameter Group 1:

6	Error code	I/O	Char(*)
---	------------	-----	---------

Optional Parameter Group 2:

7	Reset performance statistics	Input	Char(1)
---	------------------------------	-------	---------

The first and second parameter defines the program variable available for the QUSRJOB API to return the selected job information as well as the size of this variable, respectively. The third parameter specifies the format name defining the specific type of job information that you want to obtain; for the example at hand, this will be the JOBIO900 format, as explained above.

As the fourth and fifth parameter, you identify the job by job name, user name, and job number, or by the internal job identifier. The latter is a system internal identifier of any given job returned by other APIs, to let subsequent API calls locate the job faster than is possible with the qualified job name. In this case, I use the qualified job name because this is what the DSPJOBSQLI command interface provides.

For the QUSRJOB API, the standard API error data structure is an optional sixth parameter. This implies that you do not need to specify a parameter for the error data structure, unless, of course, you want to handle errors encountered by the API by means of this data structure, or unless you want to specify the seventh parameter defining whether the specified job's performance statistics should be reset. The latter parameter, however, applies only to the QUSRJOB API's format JOBIO1000, returning elapsed performance statistics. In short, optional parameters turn into required parameters if you want to specify subsequent parameters. And if optional parameters are grouped, you have to specify all parameters in the group if you specify one.

As for the optional API error data structure, the consequence of not specifying this is that the API being called signals any encountered errors by issuing an exception message to the API caller. You will therefore in your code need to cater for an exception message being returned by the API, in case you leave out the optional error data structure parameter. You have a number of options as far as evasive coding techniques are concerned, including coding a Monitor group, a CallP(e) error operation code extender followed by a %Error condition, or a Program Status Subroutine (*PSSR).

In the code provided today, I do, however, employ an API error data structure and use this to establish whether any errors were encountered when calling the QUSRJOB1 API, letting me deal with the situation accordingly. Anyway, here's the command prompt, exposing the very simple parameter interface of the DSPJOBSQLI command:

Display Job SQL Information (DSPJOBSQLI)

Type choices, press Enter.

Job name	*	Name, *
User		Name
Number		000000-999999
Output	*	*, *PRINT

You specify the qualified job name and your preference in terms of whether the command output should be displayed or printed with your job's spooled output. As always, a help text panel group is included to further explain the command and its parameters. The type and extent of job SQL information being displayed or printed depends on the type of SQL processing performed by the specified job as well as the release on which the command is run. Below I've included an example of an SQL Server job and the information returned on a release 5.4 system:

Display Job SQL Information

WYNDHAMW

23-04-11

12:07:19

Job	QSQRVR	Type
*BATCH		
User	QUSER	Status
*ACTIVE		
Number	102037	SQL server mode . . .
*CURJOB		
RDB name	WYNDHAMW	

```

Query options library . . :

SQL statement name . . . :  SQLSTATEMENT000021

SQL open cursors . . . . :  1

SQL pseudo closed cursors :  0


Cum number of SQL cursors:

Full opens . . . . . :  7

Pseudo opens . . . . . :  23


Server mode:

Connecting job . . . . :  102031/QYPSJSVR/QYPSJSVR

Connecting thread . . . :  00000044


More...
F3=Exit    F5=Refresh    F12=Cancel    F22=Display entire field
    
```

Again, the panel and all its sections and fields displayed are documented with cursor-sensitive help text that should cover any doubts about the exact interpretation of the screen content. The information displayed includes the current or most recently run SQL statement in the specified job, as in the example below. Note that the *Time started* information is shown only for currently active SQL statements:

```

                                Display Job SQL Information

WYNDHAMW                                                                23-04-11

12:07:19

Job . . . . . :  QSQSRVR      Type . . . . . :

*BATCH

User . . . . . :  QUSER      Status . . . . . :

*ACTIVE

Number . . . . . :  102037    SQL server mode . . . :

*CURJOB
    
```

Current SQL statement:

Status : *COMPLETED

CCSID : 65535

Time started :

Statement : UPDATE QMGTC.QAYPSJDT SET NAME = ?,
 OWNER = ?, EI
 M_ID = ?, CLASS = ?, CATEGORY = ?, DESCRIPTION = ?, SHARING = ?,
 STATUS = ?, V
 ERSION = ?, CREATEDDATE = ?, CHANGEDDATE = ?, DATASIZE = ?, DATA = ?
 WHERE MCK
 EY = ?

Bottom

F3=Exit F5=Refresh F12=Cancel F22=Display entire field

The current SQL statement Coded Character Set Identifier (CCSID), according to the API documentation, defines the CCSID of the current SQL statement string. During my tests on systems at release 5.4 and 6.1, however, it quickly turned out that the SQL statement string appears to be returned in the job CCSID of the job performing the QUSRJOBI API call. I have not yet had a chance to run this observation by IBM, so I've included code in the DSPJOBSQLI CPP to let you convert the SQL statement string, should this behavior change in the future. This is the part of the code that you'll need to change:

```
// The Current SQL statement appears to be returned in the job
// CCSID. Should this change in the future, you can activate
// CCSID-based conversion of the SQL statement string by removing
// the double slashes (//) for the statement below:
//
// CurSqlStmLng = CvtStrCcsId( JOBI0900.SqlStmCcsId
//                               : %Subst( JOBI0900
//                                           : JOBI0900.SqlStmOfs + 1
//                                           : JOBI0900.SqlStmLen
//                                           ));
//
```

```
// If you activate CCSID-based conversion of the SQL statement
// string, you must also comment out or remove the statement
// below:

CurSqlStmLng = %Subst( JOBI0900
                        : JOBI0900.SqlStmOfs + 1
                        : JOBI0900.SqlStmLen
                        );
```

While performing my tests, I also experienced the QUSRJOBI API returning SQL cursor names that do not conform to the regular IBM i naming convention. Some SQL cursors were named *DUMMY and had binary data appended, while other SQL cursor names were preceded by a single blank character and also had binary data appended. Here's IBM's explanation of that finding:

*DUMMY cursors exist when unique SQL statements are prepared using a statement name that isn't unique. The SQL cursor is changed to a *DUMMY cursor to allow the possibility of the cursor being re-used in the future.

Prepared SQL statements are maintained within a thread scoped internal data structure called the Prepared Statement Area (PSA). This structure is managed by the database and can be compressed. The initial threshold of the PSA is small and gradually grows through use. For an application with heavy *DUMMY cursor use, they will observe *DUMMY cursors being hard closed at each PSA compression.

The QSQBIGPSA data area control can be used to indicate that the application wants to start with a large size for the PSA threshold. By using this option, the application will skip all the PSA compressions it takes to reach a large PSA capacity.

The QSQCSRTH data area control can be used to limit the number of *DUMMY cursors. *DUMMY cursors are hard closed on PSA compression and during the cleanup processing of a PREPARE statement when the threshold is exceeded. The default threshold level is 150.

'CURSR' cursors are created when re-preparing a statement where the statement results in an internal cursor being created (cursor name is 'CURSR'). When the statement is re-prepared, the internal cursor for the previous statement is changed to a dummy cursor.

We are unable to turn the CURSR named cursor into a more easily consumed name. We precede CURSR with the blank character to insure that our internal cursor does not conflict with an actual user declared cursor. The decision to use the space leading name approach was made a long time ago. The API documentation will be updated in the next release.

In addition to the job SQL information demonstrated in the above example, other SQL-related job attributes may be displayed, depending on the type of SQL processing performed by the job in question. This includes details such as:

- SQL object name, library, and type (*PGM, *SRVPGM, *SQLPKG)
- SQL handle and descriptor counts
- SQL client registry details (application name, program ID, user ID, etc.)

- SQL interface details (interface name, type, and level)
- Server job local port number
- Client IP address and address type for server job

Most of the above SQL information was added to format JOBIO900 with release 6.1, so it will not be available when the DSPJOBSQLI command is run on release 5.4.

In order to demonstrate the practical use of the DSPJOBSQLI command—as well as the DSPJOBOPNF command presented last time—I've also included a Work with SQL Server Jobs (WRKSQLSVR) command with today's article. The WRKSQLSVR command lets you list all your system's SQL server jobs, based on user name, connecting user name, job status, and/or current user name. The SQL server jobs are named QSQRVR and identified by the server type job attribute QIBM_SQL. The QSQRVR SQL server jobs are employed by many different types of applications. I've included a list below of various application types taking advantage of the QSQRVR jobs, excerpted from an IBM Technote:

- SQL CLI applications, which enable the SQL_ATTR_SERVER_MODE environment attribute
- Native JDBC applications
- PHP applications, which use IBM DB2 extensions
- WebSphere Application Server
- IBM Directory Server
- IBM Management Central

To monitor these applications and the activity they are performing through the QSQRVR SQL server jobs, the WRKSQLSVR command comes in handy. Here's the WRKSQLSVR command prompt:

Work with SQL Server Jobs (WRKSQLSVR)

Type choices, press Enter.

User name	*ALL	Name, generic*,
*ALL...		
Connecting user	*ALL	Name, *ALL
Job status	*ACTIVE	*ACTIVE, *JOBQ,
*OUTQ...		
Current user	*ALL	Name, *ALL

Running the WRKSQLSVR command with default parameters as in the following example

```
WRKSQLSVR USER (*ALL)
           CONNUSER (*ALL)
           STATUS (*ACTIVE)
```

on my system returns the list panel displayed below:

```

                                Work with SQL Server Jobs

WYNDHAMW
                                                    23-04-11
11:58:46
User . . . : *ALL                      Connect user: *ALL

Type options, press Enter.

  2=Change   4=End   5=Work with   8=Job SQL information
10=Display job log
  11=Job open files   12=Connecting job   14=Connecting user jobs

-----
-----Current-----
-----Connecting-----
-----
Opt  Job      User      ---Status---  Job      User
Number
      QSQRVR  QDIRSRV  ACTIVE  CNDW  QDIRSRV  QDIRSRV
091475
      QSQRVR  QDIRSRV  ACTIVE  CNDW  QDIRSRV  QDIRSRV
091475
      QSQRVR  QDIRSRV  ACTIVE  CNDW  QDIRSRV  QDIRSRV
091475
      QSQRVR  QDIRSRV  ACTIVE  CNDW  QDIRSRV  QDIRSRV
091475
      QSQRVR  QDIRSRV  ACTIVE  CNDW  QDIRSRV  QDIRSRV
091475
      QSQRVR  QSECOFR  ACTIVE  CNDW  QYPSJSVR  QYPSJSVR
102031
      QSQRVR  QSECOFR  ACTIVE  CNDW  QYPSJSVR  QYPSJSVR
102031
      QSQRVR  QSECOFR  ACTIVE  CNDW  QYPSJSVR  QYPSJSVR
102031

More...
Parameters or command

===>

F3=Exit   F4=Prompt   F5=Refresh   F6=SQL commands   F9=Retrieve
F11=View 2
F12=Cancel   F21=Print list   F22=Work with active jobs
F24=More keys

```


The list panel options in addition to regular job administration tasks such as change, end, work with jobs, and display the job log also let you run the DSPJOBSQLI and DSPJOBOPNF commands for the selected job(s) using options 8 and 11. List panel options 12 and 14 let you execute the WRKJOB and WRKUSRJOB commands for the connecting job and connecting job's current user profile, respectively. For more details, please refer to the WRKSQLSVR command's and list panel's help text.

This APIs by Example includes the following sources:

```
CBX228  -- RPGLE  -- Display Job SQL Information - CPP
CBX228E -- RPGLE  -- Display Job SQL Information - UIM Exit Program
CBX228H -- PNLGRP -- Display Job SQL Information - Help
CBX228P -- PNLGRP -- Display Job SQL Information - Panel Group
CBX228X -- CMD    -- Display Job SQL Information

CBX229  -- RPGLE  -- Work with SQL Server Jobs - CCP
CBX229E -- RPGLE  -- Work with SQL Server Jobs - UIM General Exit Pgm
CBX229H -- PNLGRP -- Work with SQL Server Jobs - Help
CBX229L -- RPGLE  -- Work with SQL Server Jobs - UIM List Exit Program
CBX229P -- PNLGRP -- Work with SQL Server Jobs - Panel Group
CBX229V -- RPGLE  -- Work with SQL Server Jobs - VCP
CBX229X -- CMD    -- Work with SQL Server Jobs

CBX228M -- CLP    -- Display Job SQL Information - Build command
CBX229M -- CLP    -- Work with SQL Server Jobs - Build command
```

To create all these objects, compile and run the CBX228M and CBX229M programs, following the instructions in the source headers. You'll also find compilation instructions in the respective source headers.

Related Articles:

[APIs by Example: List Open Files API, and the Display Job Open Files Command](#)

[APIs by Example: Use a Work Management API to List Server Jobs](#)

IBM Documentation:

[DB2 for i5/OS: User-Defined Servers to the Rescue](#)

[Subsystem Configuration for SQL Server Mode Jobs](#)

[i5/OS Server Table](#)

[QSYS2.DUMP SQL CURSORS procedure](#)

This article demonstrates the following APIs:

[Retrieve Job Information \(QUSRJOBI\) API](#)

[Open List of Jobs \(QGYOLJOB\) API](#)

[Retrieve the source code for this API example.](#)

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-hidden-job-sql-information-exposed-retrieve-job-information-api>