

[print](#) | [close](#)

APIs at Work — with Jobs

[System iNEWS Magazine](#)

[Carsten Flensburg](#)

Carsten Flensburg

Fri, 09/01/2006 (All day)

[Click here](#) to download the code bundle.

To report code errors, email

[SystemiNetwork.com](mailto:carsten@systeminetwork.com)

Putting APIs to work can be challenging. APIs return data in different formats using a variety of methods, and some APIs have complex parameter lists.

Additionally, finding out exactly what an error message is trying to tell you is not always immediately obvious. But as you move along and gain experience, you will learn where to begin searching for information and how to tackle problems as they emerge.

One type of API called the Open List APIs offers a rich set of functions and a wealth of filter parameters, which let you zoom in on the exact subset of list entries that your program or utility is targeting. These APIs also have a sort information parameter, letting you sort the returned list in any sequence you could possibly want. Further, where appropriate, you can specify which fields the API should return in its return data parameter, thus the API can avoid spending CPU cycles to retrieve information for which you have no use. And finally, the Open List APIs can build the open list asynchronously in the background. This means that you can start processing the list before it's complete so that the building and the processing of the list occur in parallel, further speeding up the overall process.

IBM originally developed the Open List APIs to support client/server applications, and it put a lot of effort into making the Open List APIs as flexible and functional as possible. System i Navigator is one example of a client application that takes advantage of the Open List APIs. (One important note is that before release V5R3, the Open List APIs were all part of the Host Server (5722-SS1 option 12) of the operating system and located in library QGY. As of V5R3, the Open List APIs have been integrated in the base OS. If you are on V5R2 or earlier, you can use the Display Software Resources (DSPSFWRSC) command to verify that option 12 is installed.)

Another example is the Work with Jobs (WRKJOBS) command, which I use here to walk you through the steps involved in putting the Open List of Jobs (QGY OLJOB) API to work. (For a list of sources used to create the WRK JOBS command, see "Source Specification," below.)

Open List of Jobs

[Figure 1](#) shows the parameter list for the QGYOLJOB API, as documented at IBM's Information Center (publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qgyoljob.htm). The WRKJOBS command uses the required parameter group and optional parameter group 1. Optional parameter group 2 is relevant only if you specifically want to retrieve performance statistics data for active jobs, so I will leave those parameters out of scope for this article. [Figure 2](#) displays the QGYOLJOB API prototype with the actual API call to show the parameters and how they relate to each other. To illustrate how the parameters relate, I have grouped the 14 parameters of the QGYOLJOB API call. This grouping has nothing to do with the required and optional parameter groups in IBM's

documentation; it's simply an attempt to provide an easily comprehensible overview of the rather complex parameter list.

Let's look at the parameters at A in [Figure 2](#). The OLJBo200 parameter is the data structure ([Figure 3](#)) returned by the QGYOLJOB API. The second parameter is an expression passing the size of the OLJBo200 data structure to the API so that the API knows how much storage it can safely address. I have specified a length on the data structure definition to allow extra space for the API to return the key data (additional job attributes) defined by the parameter KeyInf (which I'll explain in a moment).

Finally, the third parameter tells the API the format name of the data structure. The format name parameter allows IBM to develop different sets of return information in terms of both extent and type, and equally important, to add new formats as dictated by future requirements.

The two parameters at B specify the field key data structure and its length, respectively. As mentioned earlier, for some return formats such as OLJBo200, which itself has a predefined format, you can further specify a number of field identifiers (field keys) to compose a variable set of job attributes to meet your requirements.

For each key specified at E, the QGYOLJOB API will return the related job attribute in a data structure ([Figure 4](#)), defining the job attribute's metadata and the offset from the beginning of the OLJBo200 data structure to the attribute value. As previously mentioned, the attribute values are returned immediately after the fixed part of the OLJBo200 data structure.

The API returns the key information data structure in parameter KeyInf (at B), allowing you to navigate through the variable part of the OLJBo200 data structure and retrieve the field key values. The following parameter tells the API exactly how much storage is available for it to use. (For a list of valid field keys and their description, go to publib.boulder.ibm.com/infocenter/iseres/v5r3/topic/apis/qusljob.htm#HDRLKEYS.)

The parameter group at C controls how the QGY OLJOB API builds the list and provides information about how to process the resultant open list. LstInf, the List Information data structure ([Figure 5](#)), is a communication area used by the API to pass list information such as whether the returned information is complete, the current and total number of records, the list handle to use when retrieving subsequent list entries, and so on. This information is intended for the API caller to control the open list retrieval and ensure that all available information is returned and that missing information is detected.

The second parameter at C is an integer; it specifies how many entries the API should return and implicitly controls whether the list is built synchronously or asynchronously. If you specify a value equal to or greater than zero, the list will be built asynchronously, and a separate server job is spawned to build the list in the background. Specifying the special value -1 will build the list synchronously, and no server job will be started. This is the value that I use for this example.

If you decide to build the list asynchronously, you can specify a value greater than 1 to retrieve more list entries in one call. I usually process one list entry at a time if no performance issues are emerging. This approach is identical to reading and processing one record at a time when processing a data file, and it makes the list retrieval process straightforward and easily adaptable to most programming requirements.

The third parameter at C, the Sort Information data structure ([Figure 6](#)), defines the order in which the returned job list is sorted. Here, you specify the number of key fields, as well as each key field's data type, data size, sort sequence, and field offset from the beginning of the return data structure

(OLJB0200). However, it is important to understand that while it's relatively simple to figure out the offset for fields in the fixed part of the data structure, this is not necessarily the case for fields in the variable part. For format OLJB0200, the field key attributes are located from offset 61 and onward.

In the first parameter at E, the sequence of the field key values reflects the order of the field key array passed to the API, but due to boundary alignment issues, the QGYOLJOB API might offset adjacent job attribute values. Therefore, the resultant offsets don't necessarily reflect the total size of preceding attributes but could be higher. To include one or more of these additional attributes in the sort configuration, I recommend that you check the actual location of the attribute in the variable part of OLJB0200. You can easily do this, for example, by adding an appropriately sized temporary field to the data structure and then using the source debugger to display the actual location of the returned data. And to make it as easy as possible, at the top of the field key array, specify the field keys to be used in the sort.

The next group of parameters (at D) is where you specify the selection criteria that should apply to the returned list. The two parameters at D and the one parameter at G specify the job selection data structure ([Figure 7](#)), the length of the data structure, and the format name, respectively. The parameter at G is optional; if it's not specified, OLJS0100 is assumed. Since I want to use OLJS0200, which offers more selection criteria, I include this parameter in the API call.

The job selection data structure is divided into two major sections. The first section defines the offset and number of each available selection criteria array; the second section is where you specify the actual selection criteria values. The second section follows immediately after the first section. For ease of use and to document the offsets, I always specify at least one array element for each criterion. For all criteria that I do not use, I simply specify zero for the values. That way, I always have a correct offset to start with for each selection criteria array, which I can then alter if I need to increase any of the array sizes.

The parameter group at E is where the additional job attributes to be returned are defined by a field key array ([Figure 8](#)). The first parameter at E simply defines to the QGYOLJOB API the number of field keys that are passed in the field key array specified for the second parameter. I define the number of array elements for the field key attribute array, and then use the built-in function %Elem () to take the array size of the field key array wherever it should be derived or is referenced. That way, if I need to alter the number of field keys, I can change this value in only one place.

Finally, the parameter at F is the API error data structure, which you use to determine whether or not the QGYOLJOB API call completed successfully. To check for a successful completion, make sure the Bytes Available field in the ERRC0100 data structure is equal to zero. If no error occurred, it is safe to proceed.

Getting at the List Entries

Once the list is initialized, use the Get Open List Entries (QGYGTLE) API to retrieve subsequent list entries. [Figure 9](#) shows the prototype and API call. The parameter list in [Figure 10](#) is merely a subset of the QGYOLJOB API parameter list, except for parameter 3, the Request Handle. This handle is a unique identifier of a list previously opened by the Open List API and ensures that the QGYGTLE API operates on the same list. The open list handle is returned by the Open List APIs as part of the List Information data structure ([Figure 5](#)).

You can omit the QGYGTLE call for a synchronously built list, but only if you choose to retrieve all available records with the initial QGYOLJOB call. To do this requires that you allocate enough storage for the API receiver variable to hold all possible entries; however, be aware that this

approach could potentially complicate the retrieval of the list entries a little. So for most situations, I recommend using the single entry retrieval approach.

I usually retrieve the list entries one by one in repeated calls to the QGYGTLE API, each time incrementing the Start record parameter by 1 until all entries have been processed or an error is encountered.

Closing Time

When the open list has been fully processed, or if the list is no longer needed, it is important to remember to explicitly close the open list. This is achieved using the Close List (QGYCLST) API ([Figure 11](#) and [Figure 12](#)), specifying the open list handle as the primary parameter. Following the call to QGYCLST, any internal storage associated with the open list is freed, and the handle specified on the call to the API is consequently no longer valid.

Presenting the List

I use a User Interface Manager (UIM) list panel group and the UIM APIs to build the WRKJOBS command's job list panel. While the details of the UIM panel and UIM API dialog are beyond the scope of this article, it still makes good sense to briefly explain the various objects involved in making WRKJOBS work:

CBX602 — Command Processing program

This program mainly performs the activities tied to displaying, or redisplaying, the WRKJOBS list panel group.

CBX602V — Validity Checking program

This program validates the existence of any specific user profile specified for either User name (USRPRF) or Current user (CURUSR) on the WRKJOBS command.

CBX602L — UIM List Exit program

One way to add list entries to a UIM list panel group is to have UIM call an exit program every time more list entries are needed to display or page down the list. This concept lets you keep all code related to initializing and extending the list in one module, as well as isolate the QGYOLJOB API handling to this module. This approach also makes any future changes to the list transparent, and having specialized modules makes them highly reusable when developing new applications and utilities.

Due to the type of dialog that UIM performs, I have separated the list building logic from the list processing logic in the program. Doing so enables the program to do both when the list is initialized or only the latter when the list is extended.

To ensure that the open list is always closed, I register an activation group exit to call the Close List API whenever a list is opened.

CBX602E — UIM General Exit program

This program prints the list, updates changed list entries, or runs the Work with Active Jobs (WRKACTJOB) command for a specific list entry.

CBX602H — Command and panel group help module

This module contains the help text for both the WRKJOBS command and the WRKJOBS list panel group.

CBX602P — WRKJOBS list panel group

This is the UIM list panel group definition for the WRKJOBS command. It defines a display list panel and a print list panel.

Worth the Challenge

The help text for the WRKJOBS command provides online documentation for the command and its parameters. If you display the general help (by pressing F1 with the cursor anywhere other than on one of the parameters and scrolling down to the last pages), you'll find several examples of how, and for what purposes, the WRKJOBS command can be used. Once you start taking advantage of the Open List APIs, you'll find they are worth the challenge.

Carsten Flensburg has been a System i programmer since 1992. His focus areas are modular system design, API programming, system integration, and communication. He enjoys the challenges involved in solving technical problems. Carsten currently works as a System i programming team leader for a European vacation rental company called Novasol, which is a part of the U.S.-based Cendant Corporation.

Find Out More

Job attributes — List of valid keys

publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qusljob.htm#HDRLKEYS

Open List of Job API

publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qgyoljob.htm

Process Open List APIs

publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/misc1b.htm

Source Specification

The Work with Jobs command is created from the following sources:

CBX602: Work with Jobs — Command Processing Program

CBX602E: Work with Jobs — UIM General Exit Program

CBX602H: Work with Jobs — Help

CBX602L: Work with Jobs — UIM List Exit Program

CBX602P: Work with Jobs — Panel Group

CBX602V: Work with Jobs — Validity Checking Program

CBX602X: Work with Jobs

Compile and run the following source to have it build the command for you (note the instructions in the source header):

CBX602M: Work with Jobs — Build command

```
CrtPgm      &UtlLib/CBX602      +
           Module( &UtlLib/CBX602 )  +
           ActGrp( *NEW )
```

```
CrtCmd      Cmd( &UtlLib/WRKJOBS)      +
            Pgm( CBX602 )              +
            SrcFile( &UtlLib/QCMDSRC )  +
            SrcMbr( CBX602X )          +
            Allow( *INTERACT *IPGM *IREXX *IMOD )  +
            AlwLmtUsr( *NO )           +
            MsgF( &UtlLib/CBX602M )    +
            HlpPnlGrp( CBX602H )       +
                                     HlpId( *CMD )
```

— C.F.

Source URL: <http://iprodeveloper.com/rpg-programming/apis-work-jobs>