

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

APIs by Example: Lock Information APIs

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 09/29/2005 (All day)

V5R2 brought a large number of new APIs to OS/400. Among these are the Retrieve Lock Information (QWCRLCKI) API and the Retrieve Lock Request Information (QWCRLRQI) API. In this installment of APIs by Example, I show you how you can put these challenging APIs to work.

The QWCRLCKI API requires thorough research to code. This is partially because it is capable of returning many different types of lock information. Depending on the input parameters, it can return any of the following:

- Object and file level locks
- File member locks
- File record locks

Using the API's filter parameter, you can limit the returned list of lock information to the following lock types:

- Lock space locks
- Job or thread locks

A lock space is an internal object used by other objects to hold object or record locks. For example, if you look at the V5R2 or later version of the ALCOBJ (Allocate Object) command, you notice that it has a SCOPE parameter that defaults to *LCKSPC. Several of the APIs introduced with V5R2 deal with this entity. I have included a list of lock space APIs at the end of this article, in case you'd like to investigate further.

Several other filters are available to reduce the size of the list that the QWCRLCKI API returns. Please check the manual if you'd like details. I've provided a link to the documentation for these APIs at the end of this article.

The output parameter of the QWCRLCKI API also offers a lot of information located in the following structures:

- Lock information list header section
- Lock information list entries
- Key information section
- Lock holder job format
- Lock holder lock space format

Included in each lock information list entry is the *Lock request handle* used to identify the lock requestor retrieved with the QWCRLRQI API. This API's output parameter includes the statement IDs and procedure name where a lock is being held. This information is returned in the form of

offsets and lengths, so you need to make a bit of extra effort to get all the information potentially available.

I have noted a few errors, contradictions, and omissions in the API documentation. These errors hardly speed up the process of learning these APIs! However, a little debugging and cross-referencing to the documentation of related APIs clears up the confusion. Anyway, the source code included with this documentation should help you get past those obstacles in no time.

The following steps are taken in the sample program to make the API calls work:

1. The QWCRLCKI input parameters are initialized based on the input to the sample program. I have included three key fields that identify attributes of the lock-holding jobs that I'd like the API to return.
2. Storage is allocated for the receiver variable so that it has a place to store the data returned. If more data is available than the parameter can hold, the program allocates additional storage and tries again.
3. A job termination procedure is registered. If the program ends unexpectedly, this procedure is called so that the storage can be properly deallocated. This can happen if the QWCRLCKI API sends back an *ESCAPE message.
4. The retrieved lock list is processed. This is done with the following steps:
 - First, the address of the *Lock list information entry* is initialized to the space in memory that the offset in the list header refers to.
 - Next, it determines whether the lock holder is a job/thread or a lock space, and the address of the corresponding job/lock information structure is set.
 - The job information key data is retrieved from the *Lock list information entry*.
 - Using the *Lock request handle* returned in the lock list information entry, the QWCRLRQI API is called to retrieve the lock requestor information.

Based on my prior experience, I'd say that the information about the program that has requested the lock is returned only for object- or member-level locks. It isn't returned for record locks.

For object- or member-level locks, the programs identified are the database manager's system jobs that implement the lock, not the application program that has requested it. Again, this is based on my experience so far.

- If more entries are available, the address of the *Lock list information entry* is advanced by the list entry size found in the list header structure.
5. When the entry list is processed, the job termination procedure is unregistered. When that's done, the program calls the job termination procedure manually to deallocate the storage for the receiver variable.

Because of space constraints, I am unable to include code snippets for all the steps that I refer to here. Instead, in the program source code, I've included comments that point out the steps.

To test the sample program that calls these APIs, I suggest that you follow the compile instructions at the top of the source members. Then, run the programs in a debug session so that you can follow the program as it executes. Here are the steps needed to do that:

1. Change the source for the CBX144T program so that it has the appropriate parameter values.
2. Compile CBX144 and CBX144T by following the instructions at the top of each source member.
3. Issue the following command and press F10:

```
STRDBG PGM(CBX144T) OPMSRC(*YES)
```

4. Call program CBX144T by typing:

```
CALL PGM(CBX144T)
```

5. Step through the program by pressing F10. When you get to the call statements, press F22 instead of F10, thereby stepping into the called programs.
6. Step through the CBX144 program by pressing F10. If this is too tedious, you can also set breakpoints by pressing F6, and then run the program until it hits a breakpoint by pressing F12.
7. Move the cursor to a variable name and press F11 to inspect its value. You can also view the contents of a variable by typing the EVAL command followed by the variable name.
8. When the program is complete, run the ENDDBG (End Debug) command to exit debug mode.

As I mentioned earlier, the QWCRLCKI and QWCRLRQI APIs return a wealth of information about locks. In addition to information that identifies the lock holder, the APIs also return granular details about the lock itself. The API documentation that I link to at the end of this article gives you all the details.

There's also a brief introduction to some of the key lock attributes and concepts at the following link: <http://publib.boulder.ibm.com/iserics/v5r2/ic2924/info/rzaks/rzakslockedobjstructure.htm>

I have included links to other, related, lock APIs at the end of this article.

The following source code is included with this article:

CBX144	Sample of calling the APIs
CBX144T	Tests CBX144 by calling it with different parameters

The source members contain instructions for how to compile these sample programs.

Important note: While testing the code included with this article, I ran into some situations where a system module would get stuck in a loop. The situation seems to arise when more than 800–1000 locks are held against a specified object, but it never seems to arise when fewer locks are held.

I have informed IBM of this problem and am waiting for a response. Until this issue is resolved, please be cautious about running the sample program, to avoid interfering with the workload in a production system. If you do get stuck in a loop, you can exit it by pressing System Request and taking Option 2.

I will let you know IBM's response as soon as I can.

Lock Space APIs:

Retrieve Lock Space Attributes (QTRXRLSA) API:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/qtrxrlsa.htm>

Retrieve Lock Space Locks (QTRXRSL) API:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/qtrxrsl.htm>

Retrieve Lock Space Record Locks (QTRXRRL) API:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/qtrxrll.htm>

Related APIs:

List Object Locks (QWCLOBJL) API:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/qwclobjl.htm>

Retrieve Job Locks (QWCRJBLK) API:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/qwcrjblk.htm>

Retrieve Job Record Locks (QDBRJBL) API:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/qdbrjbl.htm>

Retrieve Record Locks (QDBRRCDL) API:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/qdbrrcdl.htm>

This article demonstrates the following APIs:

Retrieve Lock Information (QWCRLCKI) API:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/qwcrlcki.htm>

Retrieve Lock Request Information (QWCRLRQI) API:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/qwcrlrqi.htm>

Register Call Stack Entry Termination Exit (CEERTX) API:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/CEERTX.htm>

Unregister Call Stack Entry Termination User Exit (CEEUTX) API:

<http://as400bks.rochester.ibm.com/iserics/v5r2/ic2924/info/apis/CEEUTX.htm>

Send Program Message (QMHSNDPM) API:

<http://publib.boulder.ibm.com/iserics/v5r2/ic2924/info/apis/QMHSNDPM.htm>

The following MI built-in function is demonstrated in this article:

Copy memory (_MEMMOVE):

http://publib.boulder.ibm.com/iserics/v5r1/ic2924/tstudio/tech_ref/mi/MEMMOVE.htm

You can retrieve the source code for this API example from the following link:

http://www.pentontech.com/IBMContent/Documents/article/51604_39_LockInfoApis.zip

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-lock-information-apis>