

## APIs by Example: Displaying Job Client IP Address and Job Log Information Using APIs

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 05/26/2011 (All day)



Today's APIs by Example continues the coverage of APIs that provide details or shortcuts to information not easily accessed otherwise. One example of such information is a job's current client IP address, which defines the IP address of the client currently connected to the job in question. Another example is the need to quickly spot the most recent entry in a job's job log without having to actually open the job log and navigate to the bottom of it. The former requirement is met by, for example, the Retrieve Thread Attribute (QWTRTVTA) API, and the latter is conveniently dealt with using the Open List of Job Log Messages (QGYOLJBL) API.

Employing the two mentioned APIs, I've created the Display Job IP Address (DSPJOBIPA) and the Display Job Log Message (DSPLOGMSG) CL commands. And to put the two commands, as well as the commands presented in the recent installments of APIs by Example, into a useful context, [I've further adapted the Work with Jobs \(WRKJOBS\) command offered in a System iNEWS article a few years ago](#). The WRKJOBS command also interfaces to a number of other homegrown job commands, all included with this article for your convenience. So be sure to check out the comprehensive source list and the compilation instructions at the end of this article, following your download of the source code zip file accompanying today's article.

The Retrieve Thread Attribute (QWTRTVTA) API is interesting for a couple of reasons, compared to the Retrieve Job Information (QUSRJOB) API discussed last time (see "APIs by Example: Hidden Job SQL Information Exposed by Retrieve Job Information API," article ID 65926 at SystemiNetwork.com). Whereas the QUSRJOB API returns information in grouped sets defined by each return format offered (including basic performance information, WRKACTJOB information, message logging information, active job information, etc.), the QWTRTVTA API lets you retrieve the exact job attributes that you're interested in. So the QWTRTVTA API doesn't have to spend resources and time externalizing job information that you're not going to use anyway. You simply specify a list of numeric job attribute keys identifying the specific attributes you need, and then the QWTRTVTA API returns these attributes only.

Another obvious QWTRTVTA difference compared to the QUSRJOB API, is that you can use QWTRTVTA to define the scope of the job information retrieved down to the individual thread level within the job. Here's an excerpt from the API documentation explaining the various information scope levels available, as defined by the QWTRTVTA API's thread indicator parameter included in the JIDF0100 *Job or Thread Identification Information* format:

- Information should be retrieved for the job. The value for the fields requested will be retrieved from the job. If the value requested only resides in a thread, the value for the initial thread will be returned. For example, the Current user field only resides in the thread and the initial

thread value will be returned. The returned thread identifier and the returned thread handle will be returned as hexadecimal zeros.

- Information should be retrieved for the thread specified in the thread identifier field. If the value requested only resides in a job, the value for the job containing the thread will be returned.
- Information should be retrieved for the thread that this program is currently running in. If the value requested only resides in a job, the value for the job containing the thread will be returned.
- Information should be retrieved for the initial thread of the identified job. If the value requested only resides in a job, the value for the job containing the thread will be returned.

The specific piece of information that I'll be asking the QWTRTVTA API to retrieve is associated with the job level, namely the Client IP Address. The Client IP Address is identified by attribute key 326 and is returned in a 45-byte character string, as either a formatted IPv4 or IPv6 IP address. Besides the QWTRTVTA API prototype and the standard API ERRC0100 error data structure, the code below is all that is required to retrieve a single job or thread attribute. For ease of use— and reuse—the function is encapsulated in the GetIpAddr() subprocedure:

```

**-- Get IP address:
P GetIpAddr          B
D                   Pi          45a
D  PxJobNam_q        26a  Const

**-- Local variables:
D Idx                s          10i 0
**-- Local constants:
D TIM_NO_RESET       c          '0'
D TIM_RESET          c          '1'
**-- Thread attribute format RTVT0100:

D RTVT0100           Ds          32767  Qualified
D  BytRtn            10i 0
D  BytAvl            10i 0
D  JobNam            10a
D  UsrNam            10a
D  JobNbr            6a
D                   2a
D  ThrHdl            10u 0
D  ThrId             8a
D  JobSts            10a
D                   2a
D  OfsKeyFld         10i 0
D  NbrKeyFld         10i 0
**

D KeyFld             Ds          Qualified  Based( pKeyFld )
D  FldInflLen        10i 0
D  KeyNbr            10i 0
D  DtaTyp            1a
D                   3a
D  DtaLen            10i 0
D  Data             256a

```

```

**
D JIDF0100          Ds          Qualified
D  JobNam_q          26a
D   JobNam           10a  Overlay( JobNam_q:  1 )
D   UsrNam           10a  Overlay( JobNam_q: 11 )
D   JobNbr           6a   Overlay( JobNam_q: 21 )
D  IntId             16a
D  Rsv                2a   Inz( *Allx'00' )
D  ThrInd             10i 0 Inz( -1 )
D  ThrId              8a   Inz( *Allx'00' )

/Free

JIDF0100.JobNam_q = PxJobNam_q;

RtvThrAtr( RTVT0100
           : %Size( RTVT0100 )
           : 'RTVT0100'
           : JIDF0100
           : 'JIDF0100'
           : 1
           : 326
           : TIM_NO_RESET
           : ERRRC0100
           );

If  ERRRC0100.BytAvl = *Zero;
  pKeyFld = %Addr( RTVT0100 ) + RTVT0100.OfsKeyFld;

  For  Idx = 1  to  RTVT0100.NbrKeyFld;

    If  KeyFld.KeyNbr = 326;

      Return  %Subst( KeyFld.Data: 1: KeyFld.DtaLen );
    EndIf;

  If  Idx

```

Obviously, in order to retrieve multiple attributes in one call, you'd need to define an appropriately dimensioned array of 4-byte integers and initialize this array with the numeric attribute keys identifying the requested attributes. The number of array elements would serve as the QWTRTVTA API call's sixth parameter and the attribute key array itself as the seventh API parameter. Corresponding changes would need to be made to the For-loop extracting the returned job or thread attributes, following the API call.

For more information on the QWTRTVTA API, including the job or thread identification formats as well as two other available job or thread information return formats, refer to the QWTRTVTA API documentation, for which I've provided a link at the end of this article. The Display Job IP Address (DSPJOBIPA) command is primarily based on the above GetIpAddr() function and returns the

retrieved job client IP address in an informational message sent to the job running the DSPJOBIPA command. Here's the DSPJOBIPA command prompt:

Display Job IP Address (DSPJOBIPA)

Type choices, press Enter.

Job name . . . . . *	Name, *
User . . . . .	Name
Number . . . . .	000000-999999

Command help text is provided to explain both the command and its single parameter. If you run the DSPJOBIPA command for a job associated with another user profile, you must have \*JOBCTL special authority to retrieve the client IP address information. If the interrogated job currently has established communication with a client, the resulting informational message has the following appearance:

**Job client IP address is 10.249.58.118.**

And if there's no client connection active, the following message is returned:

**No client IP address associated with job.**

The next CL command I present today, the Display Job Log Message (DSPLOGMSG) command, relies on the Open List of Job Log Messages (QGYOLJBL) API. Open List APIs is a special group of APIs following their own set of conventions. Incidentally, the article accompanying today's final command in its original version, the Work with Jobs (WRKJOBS) command, explains Open List APIs in great detail. So instead of repeating this introduction here, I've included a link to that article below (the article is titled "APIs at Work— with Jobs").

As for the specific purpose of the Open List of Job Log Messages (QGYOLJBL) API, it mainly provides a programming interface to the job log message list offered by the Display Job Log (DSPJOBLOG) command. In other words, it lets you retrieve the list of messages received by a job's job log, either starting with the newest message and going backward or starting with the oldest message and going forward, depending on your preference. The job log message information returned is divided into two sections. First, a message header section containing general message information—for example:

- message identifier
- message type
- message key
- message file name
- date and time sent

And second, a message detail section based on the same on-demand convention and format as the QWTRTVTA API, where you specify as input to the QGYOLJBL API a number of numeric key values identifying the specific message details you want to retrieve. This includes information such as

- message
- message with replacement data
- replacement data or impromptu message text
- message help
- message help with formatting characters
- default reply
- qualified sender job

The QGYOLJBL API also expects an input parameter defining a message selection information format, used by the API to navigate and select information from the job log. Here you specify, for example, the following instructions to the QGYOLJBL API:

- the qualified name of the job from which to retrieve the job log
- list direction to traverse job log (previous message/next message)
- starting message key (list direction starting point)
- maximum message length to return
- the key fields identifying message details to return

Because the objective of the Display Job Log Message (DSPLOGMSG) command is to display the most recent message sent to the specified job's job log, I set the starting message key to its maximum value, which is the hexadecimal value x'FFFFFFFF', and the list direction to \*PRV, indicating that the job log message previous to the highest possible message key value should be returned. Had I wanted to retrieve the oldest message in the job log, I would have specified a starting message key value of x'00000000' and the list direction to \*NEXT.

Note that in order to process a job log for a job associated with another user profile, you need \*JOBCTL special authority. And for jobs run by a user with \*ALLOBJ special authority, you must also have \*ALLOBJ special authority or be granted function usage to the QIBM\_ACCESS\_ALLOBJ\_JOBLOG function ID. Check out the QGYOLJBL API documentation for all the details on both security implications and functional requirements when calling this API. To get around the most obvious obstacles, you can of course also use the DSPLOGMSG command's CPP as a starting point for your own endeavors with the QGYOLBJL API. Anyway, here's what the DSPLOGMSG command prompt looks like:

```

Display Job Log Message (DSPLOGMSG)

Type choices, press Enter.
```

```

Job name . . . . . *                               Name, *
User . . . . .                               Name
Number . . . . .                               000000-999999

```

Again help text is included with the command to explain the details of the DSPLOGMSG command. The restrictions mentioned above concerning the QGYOLJBL API of course also apply to the DSPLOGMSG command. Running the DSPLOGMSG command returns the most recent message added to the specified job's job log as an informational message to the job calling the command. I ran the DSPLOGMSG command against a QSQSRVR SQL server job and got the following information in reply:

```
SERVER MODE CONNECTING JOB IS 206471/QYPSJSVR/QYPSJSVR.
```

Next, I ran the DSPLOGMSG command against a writer job in message wait (MSGW) status, and the message below appeared:

```
Operator action required on device CPHPR01 (C R).
```

Because both the DSPJOBIPA and the DSPLOGMSG command require a fully qualified job name to produce their respective output, and this in turn could imply a lot of typing of cryptic job names and job numbers as well as user profile names, it seemed like a good idea to me to create a list of jobs from which you could activate the commands using a simple list option. That also allowed for quickly issuing the commands against a selection of jobs, as well as adding some of the job-related commands I've presented earlier in this newsletter to the range of list options.

The aforementioned WRKJOBS command published in *System iNEWS* a few years ago can produce a job list based on a variety of selection criteria, so I decided to update the WRKJOBS command to include list options, in addition to a range of native job commands, for the following commands published earlier in a newsletter context:

- Display Additional Message Information Panel
- Run Job Command (RUNJOBCMD)
- Display Job Open Files (DSPJOBOPNF)
- Display Job SQL Information (DSPJOBSQLI)
- Display Job Log Message (DSPLOGMSG)
- Display Job IP Address (DSPJOBIPA)

As mentioned, I've included with this article all code necessary to build the above commands and functions and also added links to the articles explaining the commands. As for the WRKJOBS command, I also added a command parameter (ACTSTS) to support job selection based on a job's current active status, to for example enable you to list all jobs currently in a message wait state or actively running. Here's the WRKJOBS command prompt in its revised appearance:

Work with Jobs (WRKJOBS)

Type choices, press Enter.

Job name . . . . .	*ALL	Name, generic*,
*ALL...		
User name . . . . .	*ALL	Name, generic*,
*ALL...		
Job status . . . . .	*ACTIVE	*ACTIVE, *JOBQ,
*OUTQ...		
Job type . . . . .	*ALL	*ALL, *AUTO,
*BATCH...		
Current user . . . . .	*NOCHK	Name, *NOCHK
Active status . . . . .	*ALL	*ALL, *CNDW,
*DEQA, *DEQW...		
Completion status . . . . .	*ALL	*ALL, *NORMAL,
*ABNORMAL		

The Current User (CURUSR) parameter is available only when Job status (STATUS) \*ACTIVE is selected, as is the Active status (ACTSTS) parameter. Similarly, the Completion Status (COMPSTS) parameter can be entered only when job status \*OUTQ is specified. Please refer to the WRKJOBS command's help text for more information about the command and its parameters. To show an example of the job list panel, I ran the following command on my system:

```
WRKJOBS USER(Q*)
```

which in turn produced the Work with Jobs list panel below:

Work with Jobs

WYNDHAMW
15-05-11

13:43:57

Type options, press Enter.

2=Change
3=Hold
4=End
5=Work with
6=Release
7=Display

message

8=Work with spooled files
10=Display job log
12=Work with user

jobs...

13=Disconnect
14=Run job command
15=Display open files

16=Display SQL information
17=Display log message
18=Display

IP addr...

		Current			Function/	
Opt	Job	User	Job date	Type	---Status---	
Completion						
	QWCTJOBS	QSYS	11-05-11	SYS	ACTIVE	DEQW
	QWCPJOBS	QSYS	11-05-11	SYS	ACTIVE	DEQW
	QDBFSTCCOL	QSYS	11-05-11	SYS	ACTIVE	EVTW
	QSYSCOMM1	QSYS	11-05-11	SYS	ACTIVE	EVTW
	QDBSRVXR2	QSYS	11-05-11	SYS	ACTIVE	CNDW
	QQQTEMP2	QSYS	11-05-11	SYS	ACTIVE	DEQW
	QQQTEMP1	QSYS	11-05-11	SYS	ACTIVE	DEQW
	Q400FILSVR	QSYS	11-05-11	SYS	ACTIVE	DEQW
	QDBSRVXR	QSYS	11-05-11	SYS	ACTIVE	CNDW
	QFILESYS1	QSYS	11-05-11	SYS	ACTIVE	TIMW
More...						
Parameters or command						
===>						
F3=Exit    F4=Prompt    F5=Refresh    F6=Submit job    F9=Retrieve						
F11=View 2						
F12=Cancel    F21=Print list    F22=Work with active jobs    F23=More						
options						

To support the increased number of list options, I had to add function key F23 to the UIM list panel in order to enable toggling between the two sets of list options required. In the above list panel example, I included both list option sets to provide a simple overview of all supported list options. The Work with Jobs list panel and all its sections and fields displayed are documented with cursor-sensitive help text to explain all the details.

The WRKJOBS list panel's *Display message* option works on release 6.1 and later and requires a recompile of the UIM list exit program CBX232L once you get to a release later than 5.4, if the UIM list exit program originally was created on release 5.4 or earlier. The Job Information APIs did not support message queue and message key attributes for jobs in a message wait state prior to release 6.1. Compiler directives ensure that the appropriate lines of code are excluded and included, depending on the actual release when the program source is compiled.

#### **This APIs by Example includes the following sources:**

```
CBX230    -- RPGLE    -- Display Job Log Message
CBX230H   -- PNLGRP   -- Display Job Log Message - Help
```



```

CBX230X  -- CMD      -- Display Job Log Message

CBX231   -- RPGLE    -- Display Job IP Address
CBX231H  -- PNLGRP   -- Display Job IP Address - Help
CBX231X  -- CMD      -- Display Job IP Address

CBX232   -- RPGLE    -- Work with Jobs - CCP
CBX232E  -- RPGLE    -- Work with Jobs - UIM General Exit Program
CBX232H  -- PNLGRP   -- Work with Jobs - Help
CBX232L  -- RPGLE    -- Work with Jobs - UIM List Exit Program
CBX232P  -- PNLGRP   -- Work with Jobs - Panel Group
CBX232V  -- RPGLE    -- Work with Jobs - VCP
CBX232X  -- CMD      -- Work with Jobs

CBX230M  -- CLP      -- Display Job Log Message - Build command
CBX231M  -- CLP      -- Display Job IP Address - Build command
CBX232M  -- CLP      -- Work with Jobs - Build command

CBX209   -- RPGLE    -- Additional Message Information

CBX209E  -- RPGLE    -- Additional Message Information - UIM Exit
Program
CBX209H  -- PNLGRP   -- Additional Message Information - Help

CBX209P  -- PNLGRP   -- Additional Message Information - Panel Group

CBX2091  -- RPGLE    -- Display Message Queue - CPP
CBX2091H -- PNLGRP   -- Display Message Queue - Help
CBX2091V -- RPGLE    -- Display Message Queue - VCP
CBX2091X -- CMD      -- Display Message Queue

CBX209M  -- CLP      -- Additional Message Information - Build objects

CBX975H  -- PNLGRP   -- Run Job Command - Help
CBX975X  -- CMD      -- Run Job Command
CBX9751  -- RPGLE    -- Run Job Command - CPP
CBX9752  -- RPGLE    -- Run Job Command - Exit Program

CBX976   -- CLP      -- Set Job Interrupt Status Routing Program

CBX9761  -- RPGLE    -- Change Job Interrupt Status - CPP
CBX9761H -- PNLGRP   -- Change Job Interrupt Status - Help
CBX9761X -- CMD      -- Change Job Interrupt Status

CBX9762  -- RPGLE    -- Retrieve Job Interrupt Status - CPP
CBX9762H -- PNLGRP   -- Retrieve Job Interrupt Status - Help
CBX9762X -- CMD      -- Retrieve Job Interrupt Status

CBX975M  -- CLP      -- Run Job Command - Build command
CBX976M  -- CLP      -- Job Interrupt Status - Build commands

```

To create all these objects, compile and run the CBX209M, CBX230M, CBX231M, CBX229M, CBX975M, and CBX976M programs, following the instructions in the source headers. You'll also find compilation instructions in the respective source headers.

**Related Articles:**

[APIs at Work—with Jobs](#)

[APIs by Example: Message Handling APIs & Additional Message Info Support](#)

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

[APIs by Example: Hidden Job SQL Information Exposed by Retrieve Job Information API](#)

[Sending Commands to Another Job - Revisited for i5/OS V5R4](#)

[Query and Change your V5R4 Job Interrupt Status Attribute](#)

**This article demonstrates the following APIs:**

[Retrieve Thread Attribute \(QWTRTVTA\) API](#)

[Open List of Job Log Messages \(QGYOLJBL\) API](#)

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

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

**Source URL:** <http://iprodeveloper.com/rpg-programming/apis-example-displaying-job-client-ip-address-and-job-log-information-using-apis>