



How to Display the Screen of Another Interactive Job

[iPro Developer](#)

[Carsten Flensburg](#)

Carsten Flensburg

Tue, 10/23/2012 - 6:00am

APIs by Example



In this installment of APIs by Example, I'll cover two challenging programming requirements. The first relates to running a program to perform a process in another job. The second deals with communicating the 5250 data stream directly to a display device, without the intervention of a display file or panel group. For the latter, the Dynamic Screen Manager (DSM) APIs provide a robust and straightforward method for handling this requirement. An earlier approach involved manually writing 5250 data streams to a display file record format that included the DDS keyword USRDFN. The DSM APIs, however, clearly simplify that process and make display device communication much more transparent.

The reason for engaging in this interesting endeavor emerged from a request from my company's service desk. The service desk personnel ask for the ability to instantly peek at a remote interactive job's screen in order to support and guide users in need of assistance. Having a quick look over the user's shoulder is far more efficient and less error prone than having users throughout Europe spell out error messages and menu options in different languages over the phone. Looking up the interactive job via the user profile name also proved to be much speedier than trying to identify the remote terminal in question on the network. With this assignment in mind, I started investigating the options at hand to create a Display Job Screen (DSPJOBSCN) command. (For instructions on how to create DSPJOBSCN, see "How to Compile," below.)

The QWCJBITP API

Running a program in another job used to require ingenious but complex and slightly obscure methods. One approach used a message queue break-handling program in the target job; another involved the collaboration between the Start Service Job (STRSRVJOB) command and the exit program facility of the Trace Job (TRCJOB) program. Using the first method, you had to either patch a user program to system state or enlist the cooperation of the user running the target job. You could perform the latter approach without the target job user knowing that you were running a program behind the scenes. With release 5.4, IBM deactivated the TRCJOB command's exit program (EXITPGM) parameter. And in release 6.1, a new object integrity enforcement scheme removed the option to patch an object to system state.

Release 5.4, however, introduced another concept to provide system-controlled access to run programs in other jobs. The Call Job Interrupt Program is based on the following system components:

- The Call Job Interrupt Program (QWCJBITP) API lets you specify the job to interrupt and the exit program to call.
- The registration facility's QIBM_QWC_JOBITPPGM exit point requires you to add the specified exit program to this exit point before the QWCJBITP API call.
- The system value Allow Jobs to be Interrupted (QALWJOBITP) defines one of three interruptible states. The default value 0 (zero) disallows job interruption altogether. A value of 1 allows job interruption, but all new jobs default to uninterruptible. A value of 2 also permits job interruption, but all new jobs default to interruptible.
- When the QALWJOBITP system value is set to 1, using the Change Job Interrupt Status (QWCCJITP) API lets you change the job's initial uninterruptible status to interruptible from within the job itself. You can easily overcome this obstacle in an interactive job given some cooperation from the job user; batch jobs, however, are out of reach.

Note that immediately before running the exit program, the system will swap the user profile of the target job's

initial thread to the user profile of the QWCJBITP API's caller. After the exit program has completed (due to normal exit conditions or to an error), the user profile of the target job's initial thread is swapped back to the user profile that was in effect before the exit program call. This implies that the exit program is run in the target job with the authority and privileges of the user profile calling the QWCJBITP API.

Let's go through the QWCJBITP API parameter list in Figure 1.

Figure 1: The QWCJBITP API parameter list

```
Call Job Interrupt Program (QWCJBITP) API
```

Required Parameter Group:

1 Input variable	Input	Char(*)
2 Input format name	Input	Char(8)
3 Error code	I/O	Char(*)

The input variable defines a data structure that conveys the information necessary for the QWCJBITP API to locate the exit program to run, the target job to interrupt, and up to 2,000 bytes of program data, which will be provided as the primary input parameter to the exit program. The input format name identifies the format of the input variable (Figure 2)—currently, only format JITPo100 is available.

Figure 2: Input variable format JITPo100

Offset			
Dec	Hex	Type	Field
0	0	Char(10)	Program name
10	A	Char(10)	Program library
20	14	Char(10)	Target job name
30	1E	Char(10)	Target job user
40	28	Char(6)	Target job number
46	2E	Char(2)	Reserved
48	30	Binary(4)	Offset to program data
52	34	Binary(4)	Length of program data
		Char(*)	Program data
		Char(*)	Reserved

The program name and library identify the program that the QWCJBITP API will call in the interrupted job. For the call to succeed, the specified program must have been registered with the QIBM_QWC_JOBTPPGM exit point. The target job name, user, and number identify the job to interrupt and subsequently call the exit program. The program data is submitted to the exit program as the exit program's first parameter. The program data can have a maximum length of 2,000 bytes. The exit program's second parameter is a four-byte integer that holds the actual length of the program data.

The DSM APIs

At this point, you have the requisites to run a program in another job. Now you need to find out how to retrieve the current screen in an interactive job. This is where the DSM APIs come in handy. The DSM APIs offer a set of screen I/O functions that can create and manipulate screens dynamically. Usually, you define screens by coding display files or panel groups in DDS or the user interface manager (UIM), respectively. With DSM, you can use a series of calls to the DSM APIs to specify and control screen appearance on the fly. Because the DSM interfaces are bindable, they're accessible to ILE programs only. The DSM APIs are divided into three functional groups, as described in the IBM i5/OS Information Center:

- Low-level services provide a direct interface to the 5250 data stream commands. You use these APIs to query and manipulate the state of the screen; to create, query, and manipulate input and command buffers used to interact with the screen; and to define fields and write data to the screen.
- Window services let you create, delete, move, and resize windows as well as manage multiple windows during a session.
- Session services provide a general scrolling interface that you can use to create, query, and manipulate sessions, and to perform input and output operations to sessions.

In general, the DSM APIs perform their duties in either direct or indirect operation mode. In direct mode, you use a command buffer that accumulates and stores a group of requested screen operations; the accumulated

operations can then be written to the screen in a single I/O operation.

For a more detailed and comprehensive discussion of the DSM APIs, as well as useful programming examples, check out the links in "Find Out More," below. Here, you'll find articles by Gene Gaunt, Craig Pelkie, and Scott Klement, as well as links to the online version of the *5494 Functions Reference* (SC30-3533-04), which includes a chapter titled "Topic 15. Display Data Streams" that documents many of the 5250 data stream commands that the DSM APIs employ. The *5250 Functions Reference* (SA21-9247-06) exclusively focuses on the 5250 data stream (this manual is available only in hardcopy from the IBM Publication Center, to which a link is also included).

Now on to the steps involved in retrieving the current screen in a remote, interactive job and displaying the retrieved screen in the current job:

1. Check whether a communication data queue exists, and create one if necessary.
2. Initialize the QWCJBITP API's JITPo100 parameter data structure.
3. Initialize the API program data structure.
4. Create a unique key for the data queue *put* and *reply* entries.
5. Move the data queue entry to the utility data queue and wait for a reply.
6. Call the QWCJBITP API and pass the JITPo100 parameter data structure.

At this point, the target job takes control:

7. Get the data queue entry by key based on the program data structure information.
8. Retrieve the current screen image and other screen information.
9. Update the return data structure with the retrieved screen information.
10. Move the data queue entry with the reply key to the utility data queue and end.

Now, the current job regains control:

11. Get the data queue entry by reply key and process the returned data.
12. Clear the current screen.
13. Restore the retrieved screen and cursor position.
14. Wait for the function key to be pressed. The F3 key ends the program, and F5 repeats steps 2 to 14.

The DSM APIs perform all of these screen I/O functions. Usually, the screen I/O operations occur within the same job, so all I/O buffers are stored internally in the job and referenced only through a buffer handle that's returned from one DSM API and subsequently passed to the next DSM API. In this case, however, I need to retrieve the actual screen buffer data. A buffer handle is valid only within the same job—if I can't access the buffer data, I can't pass it back to the job running the DSPJOBSCN command.

To get around this obstacle, I use the two DSM APIs provided for situations in which an I/O operation isn't directly supported by DSM or lacks support suitable for the purpose at hand, as is the case here. The two DSM APIs—QsnPutInpCmd and QsnPutOutCmd—let you perform I/O operations by using the appropriate 5250 data stream commands directly. In the given context, this amounts to the 5250 commands SAVE SCREEN and RESTORE SCREEN. Further, the SAVE SCREEN command implicitly includes the RESTORE SCREEN command and escape sequence in the returned command buffer. You can therefore feed the command buffer directly to the QsnPutOutCmd API to restore the retrieved screen.

By reviewing the CBX2561 and CBX2562 sources or running a source debug session while the programs are executing, you can see how the screen dialog unfolds in detail. To debug the CBX2562 program, start a debug session against this program in the target job, and then run the DSPJOBSCN command against the target job from another session. I've updated the *Work with Jobs* (WRKJOBS) command presented in earlier articles to include the list option 15=Display job screen, which makes it a bit easier to locate relevant jobs and issue the DSPJOBSCN command:

```
WRKJOBS JOB (*ALL)
USER (*ALL)
STATUS (*ACTIVE)
JOBTYPE (*INTER)
```

To further narrow the list, you can stipulate a specific or generic job or username. If the system value

QALWJOBITP is set to 1 so that a job defaults to uninterruptible, you can change a job's interrupt status to interruptible by running the following Change Job Interrupt Status Attribute (CHGJOBITPS) command in that job:

CHGJOBITS P TIPSTS (*ALWITP)

If the system value QALWJOBITP is set to 2, all jobs are interruptible by default. In that case, you can use the CHGJOBITS command to disallow job interruption in a particular job. Using routing entries lets you globally control the default interrupt status attribute of jobs beyond the granularity supported by the QALWJOBITP system value. (For details on how to do this, see the related articles in "Find Out More.")

You can see the DSPJOBSCN command prompt in Figure 3.

Figure 3: Display Job Screen (DSPJOBSCN) command prompt

Type choices, press Enter.

Job name	Name
User	Name
Number	000000-999999
Request time-out	Seconds, 5-3600

The command accepts the qualified name of the job for which to list the current screen, as well as a second command parameter that defines how many seconds the job interrupt process has before it times out.

Figure 4 shows an example of the retrieved job screen revealing the system main menu as the current job screen for the target job.

Figure 4: Display Job Screen display panel

MAIN IBM i Main Menu System: WYNDHAMW

Select one of the following:

1. User tasks
2. Office tasks
3. General system tasks
4. Files, libraries, and folders
5. Programming
6. Communications
7. Define or change the system
8. Problem handling
9. Display a menu
10. Information Assistant options
11. IBM i Access tasks

90. Sign off

Selection or command
====>

F3=Exit F4=Prompt F9=Retrieve F12=Cancel F13=Information Assistant
F23=Set initial menu

(C) COPYRIGHT IBM CORP. 1980 - 2000

F3=Exit F5=Refresh

The function key texts F3=Exit and F5=Refresh reflect the option to either cancel the display job screen command or refresh the current screen, respectively. To toggle the function key texts on and off, use the F2 key. The target job screen cursor position is also retrieved and added to the displayed screen copy.

Running Commands in Other Jobs

Using the same basic techniques I present in this article, you can employ the job interrupt facility to run commands in other jobs. For a discussion on how to implement the Run Job Command (RUNJOBCMD), read the article "Sending Commands to Another Job – Revisited for i5/OS V5R4" listed in "Find Out More." By the way, the revised version of the WRKJOBS command also supports RUNJOBCMD as option 14.

How to Compile

Below are instructions on how to create the Display Job Screen command. The following sources are included with the code bundle associated with this article:

- CBX256H–PNLGRP: Display Job Screen–Help
- CBX256V–RPGL: Display Job Screen–VCP
- CBX256X–CMD: Display Job Screen
- CBX2561–RPGL: Display Job Screen–CPP
- CBX2562–RPGL: Display Job Screen–Exit Program
- CBX256M–CLP: Display Job Screen–Build command

To create all of these command objects, compile and run the CBX256M CL program, and follow the instructions in the source header. Note that it's critical that you read and follow the instructions in the CBX256M CL source header before you compile and run the program. You'll also find compilation instructions in the respective source headers of the individual sources.

The revised version of the *Work with Jobs* command includes the following sources:

- CBX232–RPGL: Work with Jobs–CCP
- CBX232E–RPGL: Work with Jobs–UIM General Exit Program
- CBX232H–PNLGRP: Work with Jobs–Help
- CBX232L–RPGL: Work with Jobs–UIM List Exit Program
- CBX232P–PNLGRP: Work with Jobs–Panel Group
- CBX232V–RPGL: Work with Jobs–VCP
- CBX232X–CMD: Work with Jobs
- CBX232M–CLP: Work with Jobs–Build command

To create these command objects, compile and run the CBX232M CL program, and follow the instructions in the source header. You can download the Change Job Interrupt Status (CHGJOBITS) command by following the link for the article "Query and Change your V5R4 Job Interrupt Status Attribute" in "Find Out More."

Find Out More

["5250 Data Stream Programming"](#)

[5494 Functions Reference \(SC30-3533-04\)–"Topic 15. Display Data Streams"](#)

[5494 Functions Reference–PDF version](#)

["Color by Numbers with DSM"](#)

["Dynamic Menu Bars with DSM"](#)

[Dynamic Screen Manager Message Subfile \(forum thread\)](#)

["Getting Started with DSM Input Fields"](#)

[IBM Publication Center](#)

IBM I 7.1 Information Center documentation

[Allow Jobs to be Interrupted \(QALWJOBITP\) System Value](#)

[Call Job Interrupt Program \(QWCJBITP\) API](#)

[Call Job Interrupt Program Exit Program](#)

[Change Job Interrupt Status \(QWCCJITP\) API](#)

[Clear Buffer \(OsnClrBuf\) API](#)

[Clear Screen \(QsnClrScr\) API](#)

[Create Command Buffer \(QsnCrtCmdBuf\) API](#)

[Create Input Buffer \(QsnCrtInpBuf\) API](#)

[Delete Buffer \(QsnDltBuf\) API](#)

[Dynamic Screen Manager APIs](#)

[Get Cursor Address \(QsnGetCsrAdr\) API](#)

[Put Input Command \(QsnPutInpCmd\) API](#)

[Put Output Command \(QsnPutOutCmd\) API](#)

[Query Display Mode Support \(QsnQryModSup\) API](#)

[Read Modified Fields \(QsnReadMDT\) API](#)

[Retrieve Field Information \(QsnRtvFldInf\) API](#)

[Retrieve Pointer to Data in Input Buffer \(QsnRtvDta\) API](#)

[Retrieve Screen Dimensions \(QsnRtvScrDim\) API](#)

[Retrieve Display Mode \(QsnRtvMod\) API](#)

[Set Cursor Address \(QsnSetCsrAdr\) API](#)

[Write Data \(QsnWrtDta\) API](#)

Articles at iProDeveloper.com

["APIs by Example: Displaying Job Client IP Address and Job Log Information Using APIs"](#)

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

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

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

["Automatic Screen Prints"](#)

["Grab Data from the Screen"](#)

["Grab Data from the Screen, Part 2"](#)

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

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

["Variable Positions and Field Lengths with the Dynamic Screen Manager"](#)

Source URL: <http://iprodeveloper.com/rpg-programming/how-display-screen-another-interactive-job>