

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

APIs by Example: Data Queue APIs and CL Commands

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 11/16/2006 (All day)

Last time, I showed you how to make API-provided functions available through CL command interfaces, and today I offer another couple of examples of doing that. However, this time I use the APIs directly as the command's CPP, and I demonstrate the techniques required to accomplish that.

The focus of this and upcoming issues of APIs by Example will, however, be the Data Queue APIs. Data Queues provide a speedy, versatile, and functional program-to-program communication method and also provide a perfect coupling and transaction layer in modern application design models, aiming at separating the data processing and business logic from the presentation layer.

To elaborate a bit on my statement, let me explain some of the application design objectives that can be achieved with data queues:

1. Speed — The receiving program will already have been activated at the point at which it is waiting on the data queue for messages to arrive. The call level has been established, initiation activities performed, and the data queue has been located and authority has been verified.
2. Scalability — More than one process or job can be waiting on the same inbound data queue, so that as many messages can be processed in parallel as server jobs have been submitted. The sending job includes a reply key to uniquely identify the request, and that key is used when the request processor returns the reply message to an outbound keyed data queue. Specifying that reply key on the receive data queue API call ensures that the receiving job gets the correct reply from the outbound data queue.
3. Threading — Using a keyed inbound data queue, you can have different server jobs waiting against the same data queue for different types of request to process. One client could issue three different request messages, have them processed in parallel by three different server jobs, and then continue processing when all reply messages have been returned.
4. Distribution — One request could also be distributed to different applications or servers by sending the same request to different data queues and then having the reply messages consolidated into a single reply upon return.

I should add to point number one that if you are chasing the last milliseconds, only user queues are more efficient in performing queue services than data queues. This speed, however, has its price in that user queues are a bit more complex and difficult to program. At the end of this article, I provide a link to an article that explains and demonstrates how to program user queues, in case you're interested in pursuing that option.

I also added a number of other links to data queue articles, explaining and/or demonstrating how to exploit data queues in application development. I'll go into more details on this perspective in upcoming issues of this column, but I devote the rest of today's issue to adding three data queue

commands currently missing in the System i's native collection. If you display the Data Queue Command menu, you get a choice of the three currently available data queue commands. Running the command GO CMDDTAQ presents the following supply:

```

CMDDTAQ                                Data Queue Commands
Select one of the following:
Commands
  1. Create Data Queue                  CRTDTAQ
  2. Delete Data Queue                 DLTDTAQ
  3. Work with Data Queues             WRKDTAQ

```

Although option 3, Work with Data Queues, at first looks promising, it actually provides only an interface to creating data queues, deleting data queues, and changing the object description of data queues. Further investigation also reveals that an otherwise very useful command, Create Duplicate Object (CRTDUPOBJ), actually does not support object type *DTAQ.

When using data queues in your application design and development, you might therefore sometimes find yourself missing a data queue tool or two, to help in testing and verifying the process. An example of one obvious omission is the Display Data Queue Description (DSPDTAQD) command, but because IBM provides the equivalent Retrieve Data Queue Description (QMHQRDQD) API, building a DSPDTAQD command is an obstacle that we can overcome:

```

                                Display Data Queue Description (DSPDTAQD)
Type choices, press Enter.
Data queue . . . . . Name
Library . . . . . *LIBL Name, *LIBL, *CURLIB
Output . . . . . * *, *PRINT

```

The QMHQRDQD API is a pretty straightforward API to call, so I do not go into detail about that exercise here, but I leave it as an option to check out the accompanying code if this part of the challenge is of interest to you. Running the command DSPDTAQD QSYSDTAQ on my system results in the display of the following panel:

```

                                Display Data Queue Description                                WYNDHAMW
                                                                 12-11-06 08:27:10
Data queue . . . . . : QSYSDTAQ      Type . . . . . : *STD
Library . . . . . : QSYS
Maximum entry length . . . . . : 144
Force to auxiliary storage . . . : *NO
Sequence . . . . . : *KEYED
Key length . . . . . : 4
Include sender ID . . . . . : *NO
Automatic reclaim . . . . . : *YES
Queue size:
Maximum number of entries . . . : *MAX2GB
Initial number of entries . . . : 16
Queue entries:
Current number . . . . . : 4049

```

```

Current allocated . . . . . : 4112
Maximum allowed . . . . . : 9586813

Text 'description' . . . . . : Exit point: QIBM_QWT_JOBNOT
IFY -format: NTFY0100

Bottom

F3=Exit   F5=Refresh   F12=Cancel

```

The information presented varies depending on the type of data queue specified, standard or DDM, respectively. Full online and cursor-sensitive help text is provided for both command and display panel.

To help you test the DSPDTAQD command and data queues in general, I also provide the Send Data Queue Entry (SNDDTAQE) and Clear Data Queue (CLRDTAQ) commands. As I mentioned earlier, these commands are defined so that the equivalent APIs can be called directly as the CPP for the commands. Here's the full SNDDTAQE command prompt:

```

Send Data Queue Entry (SNDDTAQE)
Type choices, press Enter.
Data queue . . . . . Name
Library . . . . . *LIBL Name, *LIBL, *CURLIB
Entry data length . . . . . 1-3000
Entry data . . . . .
...
Key length . . . . . *NONE 1-256, *NONE
Key data . . . . . *NONE
Additional Parameters
Asynchronous request . . . . *NO *NO, *YES
Data from journal entry . . *NO *NO, *YES

F3=Exit   F4=Prompt   F5=Refresh   F10=Additional parameters
F12=Cancel F13=How to use this display F24=More keys

```

And the CLRDTAQ command prompt looks like this with all parameters prompted:

```

Clear Data Queue (CLRDTAQ)
Type choices, press Enter.
Data queue . . . . . Name
Library . . . . . *LIBL Name, *LIBL, *CURLIB
Additional Parameters
Key order . . . . . *NONE *NONE, *GT, *LT, *NE...
Key length . . . . . *NONE 1-256, *NONE
Key data . . . . . *NONE

F3=Exit   F4=Prompt   F5=Refresh   F10=Additional parameters
F12=Cancel F13=How to use this display F24=More keys

```

I also include help text panel groups to document the commands and their parameters. As I mentioned last time, most of the documentation is copied from the API documentation, so this is a quick and easy way to provide command information right at hand.

To use an API as a CPP directly, a number of techniques are required:

1. All API parameters should be specified as command parameters, including optional and omissible API parameters. Specify the default values for the API parameters as the defaults for the command parameters.
2. Use the command definition Prompt Control *PMTRQS keyword to hide less interesting or seldom used command parameters, as in the following example. Then use F10 to display these hidden parameters, if necessary.

Parm	ASYNCH	*Char	10	+
	Rstd(*YES)			+
	Dft(*NO)			+
	SpcVal((*NO) (*YES))			+
	Expr(*YES)			+
	PmtCtl(*PMTRQS)			+
	Prompt('Asynchronous request')			

3. Specify command special values with substitution for API special values, as for example *NOMAX instead of -1, or *NONE instead of 0, as in the following example. When the operator specifies *NONE for the KEYLEN parameter, 0 is sent to the CPP.

Parm	KEYLEN	*Dec	(3 0)	+
	Dft(*NONE)			+
	SpcVal((*NONE 0))			+
	Range(1 256)			+
	Expr(*YES)			+
	Prompt('Key length')			

4. Specify a command parameter constant for the API error data structure. The constant must have a value of x'00000000', because this signals the API to send exception messages in case of an error. See the next example. Constant parameters are not displayed on the command prompt, only passed on to the CPP.

Parm	ERRC0100	*Char	4	+
	Constant(x'00000000')			

Following these guidelines enables you to specify an API directly as a CPP and makes it easier for the command users to understand what the various special values actually mean.

After you've successfully created all three commands, I suggest that you create one or more test data queues in a library that doesn't interfere with production activities. Using the SNDDTAQE command, you can then add test entries to your data queue(s) and display the effect on the data queue with the DSPDTAQD command. As for the latter, notice that you can use F5 to update the panel without having to rerun the DSPDTAQD command. Finally, use the CLRDTAQ command to remove entries from the data queue, either by key or as an all-entry clear.

In upcoming issues of APIs by Example, I will get into more detail about how to configure and use data queues, and I'll also provide some more of the missing data queue commands. In the meantime, I provide some links for you at the end of this article, allowing you to read more about data queues on your own.

This APIs by Example includes the following sources:

```
CBX165    -- Display Data Queue Description - CPP
CBX165H   -- Display Data Queue Description - Help
CBX165P   -- Display Data Queue Description - Panel Group
CBX165V   -- Display Data Queue Description - VCP
CBX165X   -- Display Data Queue Description

CBX1652H  -- Clear Data Queue - Help
CBX1652X  -- Clear Data Queue
CBX1653H  -- Send Data Queue Entry - Help
CBX1653X  -- Send Data Queue Entry

CBX165M   -- Data Queue Commands - Build commands
```

To create all these objects, compile and run CBX165M. Compilation instructions are in the source headers, as usual.

Data queue articles and utility examples:

Using Data Queue APIs:

<http://www.systeminetwork.com/article.cfm?id=1349>

Using Data Queues in RPG/IV:

<http://www.systeminetwork.com/article.cfm?id=3314>

Data Queues: A PC-to-iSeries Quick Link:

<http://www.systeminetwork.com/article.cfm?id=15842>

Automatically Send Messages When Jobs Start, End or Are Queued:

<http://www.systeminetwork.com/article.cfm?id=16738>

Automatic Notification When Spool Files Are Created:

<http://www.systeminetwork.com/article.cfm?id=18244>

APIs by Example: Realtime Job Monitor:

<http://www.systeminetwork.com/provipcenter/index.cfm?fuseaction=ShowNewsletterIssue&ID=19252>

An Introduction to Asynchronous Messaging:

<http://www.systeminetwork.com/article.cfm?id=20523>

User queue articles:

APIs by Example: Creating a User Queue:

<http://www.systeminetwork.com/article.cfm?id=17972>

APIs by Example: User Queues, Part 2:

<http://www.systeminetwork.com/article.cfm?id=18097>

This article demonstrates the following data queue APIs:

Retrieve Data Queue Description (QMHQRDQD) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qmhqrdqd.htm>

Send Date Queue (QSNDDTAQ) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qsnddtaq.htm>

Clear Data Queue (QCLRDTAQ) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qclrdaq.htm>

All data queue APIs are documented here:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/obj2.htm>

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

http://www.pentontech.com/IBMContent/Documents/article/53542_143_DataQueue1.zip

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-data-queue-apis-and-cl-commands>