

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

## APIs by Example: Data Queue APIs and CL Commands, Part 2

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 12/14/2006 (All day)

As I discussed in part 1 of this series (a link to part 1 is at end of this article), data queues offer a brilliant method of program-to-program communication. The Distributed Data Management (DDM) type of data queues offer a way to extend this capacity to include system-to-system communication.

In the past, DDM data queues, originally based on the SNA facility, were often challenging to configure, especially as TCP/IP became the preferred communication protocol. But with the introduction of relational database (RDB) –based identification of a remote system, this challenge was dramatically reduced.

In this article, I discuss DDM data queues, and I also add to my collection of "missing" data queue CL commands, presenting the Copy Data Queue Description (CPYDTAQD) command.

Let's start with DDM data queues. DDM data queues offer a convenient way to let two applications on different System i computers communicate with each other.

One scenario could be two companies that want to integrate their business applications, because company A just bought company B. Another scenario could be the desire to integrate a business application running on one system with a new finance application running on yet another system. And a third example could be a web front-end application needing realtime access to a back-end business application running on a different server.

As I discussed in part one, using data queues as a communication layer between programs offers a number of advantages in itself, and these advantages also apply to DDM data queues. In this case, you also gain the portability aspect: Applications on different servers communicating through remote (DDM) data queues can easily be consolidated on one server. Restore the application and replace the remote data queue with a local one, and you're done. The other way around also works, of course.

Using the \*RDB option to identify the remote location is the easiest way to configure DDM data queues in today's TCP/IP networks. The following steps are involved in setting up DDM data queues:

1. On the system on which you want to configure the DDM data queue, use the Work with Relational Database Directory Entry (WRKRDBDIRE) command to find the relational database name of the remote system. If it is not already there, use the Add Relational Database Directory Entry (ADDRDBDIRE) command to add it. Follow the help text of the ADDRDBDIRE command or the IBM Technical Document (see the link for it at the end of this article) to create the RDB directory entry.
2. On the CRTDTAQ command, specify the RDB name of the server in step 1. I keep the names of the local and remote data queues the same. In the following example, the RDB server name is CPHDB01, and the data queue name is WEBINBDQ in library QGPL:

```
CRTDTAQ DTAQ (QGPL/WEBINBDQ)
      TYPE (*DDM)
      RMTDTAQ (QGPL/WEBINBDQ)
      RMTLOCNAME (*RDB)
      RDB (CPHDB01)
      TEXT('Web application inbound data queue')
```

3. To enable the user profile that puts or gets data queue entries on or from the preceding DDM data queue to successfully connect to the remote system, the user profile and password need to be identical on the two

systems. If this is infeasible, you can use a Server Authentication Entry to store the remote user profile name and password. In the next example, I use the current user profile as the local user profile and the RDB entry name as the server name to achieve a successful connection with the remote user profile CPHWEBUSR:

```
ADDSVRAUTE USRPRF(*CURRENT)
           SERVER(CPHDB01)
           USRID(CPHWEBUSR)
           PASSWORD(remote password)
```

Before adding server authentication entries on your system, please refer to "Source System Security in a TCP/IP Network — Server Authentication" (a link is at the end of this article), which includes a thorough discussion of the Retain Server Security (QRETSVRSEC) system value.

4. After putting a data queue entry on the WEBINBDQ on the local system, the result is a data queue entry showing in the WEBINBDQ on the remote system, provided that the data queue exists on the remote system and that the connection to the remote system was successful. Otherwise, the Send Data Queue (QSNDDTAQ) API returns an error.

Performing the same drill for the WEBOUDDQ data queue on the two systems should lead to the existence of an inbound data queue and an outbound data queue on both systems. I usually keep the local data queue on the system on which the data queue entries are being processed. So I put data queue entries on a DDM data queue, and I get data queue entries from a local data queue. That way, when sending the data queue entry, I know already whether the communication was successful, and I also consistently know where to perform recovery and error processing. In some of my data queue-driven applications, I also use the Retrieve Data Queue Message (QMHRDQM) API to monitor the content of the inbound data queue, but the QMHRDQM API unfortunately does not support DDM data queues.

At this point, you're ready to set up the applications that are going to communicate with each other through the data queues created in the preceding examples. I'll provide an example of such a setup in an upcoming installment of this article series.

For now, let me close the DDM data queue topic with a little hint for when you're trying to find DDM data queues on your system: A quick way to identify DDM type data queues is to run the Work with Objects (WRKOBJ) command. The resulting panel includes the object attribute column, which has a value of DDMDTAQUE for all DDM type data queues. Try running the following command:

```
WRKOBJ OBJ(*ALL) OBJTYPE(*DTAQ)
```

Look for all data queue objects that have the DDMDTAQUE object attribute. You can use the Display Data Queue Description (DSPDTAQD) command that I provided in part 1 to display all data queue attributes, regardless of type.

The next item on the agenda for today is the CPYDTAQD command. Here's what the command prompt looks like:

```
Copy Data Queue Description (CPYDTAQD)

Type choices, press Enter.
From data queue . . . . . Name
  Library . . . . . *LIBL Name, *LIBL, *CURLIB
To data queue . . . . . Name
  Library . . . . . *CURLIB Name, *CURLIB
Type . . . . . *STD *STD, *DDM
Maximum entry length . . . . . 1-64512
Force to auxiliary storage . . *NO *NO, *YES
Sequence . . . . . *FIFO *FIFO, *LIFO, *KEYED
Key length . . . . . 1-256
Include sender ID . . . . . *NO *NO, *YES
```

```

Queue size:
Maximum number of entries . *MAX16MB      Number, *MAX16MB, *MAX2GB
Initial number of entries . 16             Number
Automatic reclaim . . . . . *NO           *NO, *YES
Remote data queue . . . . .              Name
Library . . . . . *LIBL                   Name, *LIBL, *CURLIB
Remote location . . . . .                Name, *RDB
Relational database . . . . .
APPC device description . . . *LOC         Name, *LOC
Local location . . . . . *LOC             Name, *LOC, *NETATR
Mode . . . . . *NETATR                   Name, *NETATR
Remote network identifier . . *LOC        Name, *LOC, *NETATR
Text 'description' . . . . . *BLANK
Additional Parameters
Authority . . . . . *LIBCRTAUT           Name, *LIBCRTAUT...

```

Because the retrieval of the *from* data queue's attributes is performed by the CPYDTAQD command's prompt override program (POP), the copy function is performed when the command is entered, be it on a command line, in a CLP source member, or on the Submit Job (SBMJOB) command prompt. Any subsequent replacement of the *from* data queue involving new queue attributes or any library list look-up resulting in localization of another data queue with the specified queue name will not therefore be reflected in future executions of the CPYDTAQD command.

This behavior ensures that the data queue attributes are always predictable in relation to the Send or Receive Data Queue API calls. For the same reason, if you need to create data queues on the fly in a program, the CRTDTAQ command should be fully coded in the program (i.e., all crucial attributes should be explicitly specified on the CRTDTAQ command). Don't rely on the CRTDTAQ command's default parameter settings, because they could be subject to change in the future by an unforeseen use of the Change Command Default (CHGCMDDFT) command.

The CPYDTAQD command includes full online help documentation. Because the CPYDTAQD command in essence is a CRTDTAQ command in disguise, I borrowed most of the help text from the CRTDTAQ command's help text panel group. Such adoption can in some instances be an elegant shortcut to provide online documentation, so here's a brief recipe for how to achieve that:

1. Identify the Help Panel Group of the source command, the HLPPNLGRP attribute, by running the following command :

```
DSPCMD CMD(CRTDTAQ)
```

2. Specify the found panel group on an :IMPORT tag in your help panel group, in this case the QHMHCMDCMD panel group:

```
IMPORT PNLGRP='QHMHCMDCMD' NAME='* '.
```

3. The NAME= attribute of the :IMPORT tag must be either the name of the command parameter whose help text you want to import or the special value '\*'. The latter can be specified only for one :IMPORT tag per panel group and causes the User Interface Manager (UIM) to look in the specified PNLGRP= for all unresolved :IMHELP tags. In this case, I'm importing from only one panel group, so using this approach is the easiest way. Otherwise, I would have to specify an :IMPORT tag for each imported command parameter help text.
4. For each imported help text, specify the :IMHELP tag, as in the following example for the data queue type parameter:

```
:HELP NAME='CPYDTAQD/TYPE'.Type (TYPE) - Help
:IMHELP NAME='CRTDTAQ/TYPE'.
:EHELP.
```

5. Create the panel group and be careful to verify that all imported command parameter help texts have successfully been resolved:

```

Type (TYPE) - Help
Specifies the type of data queue to be created. A
standard data queue or a distributed data management (DDM)
data queue can be created.

```

#### **\*STD**

A standard data queue is created. The MAXLEN parameter is required with the use of this value.

#### **\*DDM**

A DDM data queue is created. This value requires the name of the remote data queue accessed (RMTDTAQ parameter) and the name of the remote (target) system that the data queue is located on (RMTLOCNAME parameter).

That's it for now, but in the next installment of APIs by Example, I will continue my coverage of data queue APIs and data queue CL commands, as well as data queue programming and use.

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

```

CBX166  -- Copy Data Queue Description - CPP
CBX166H -- Copy Data Queue Description - Help
CBX166O -- Copy Data Queue Description - POP
CBX166V -- Copy Data Queue Description - VCP
CBX166X -- Copy Data Queue Description

CBX166M -- Copy Data Queue Description - Build commands

```

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

### **IBM Technical Documents:**

Configuring DDM Data Queue Support over TCP/IP:

[https://www-912.ibm.com/s\\_dir/slkbase.NSF/1ac66549a21402188625680b0002037e/e95c5072635554dd86256e980068aada?OpenDocument](https://www-912.ibm.com/s_dir/slkbase.NSF/1ac66549a21402188625680b0002037e/e95c5072635554dd86256e980068aada?OpenDocument)

Assigning a Default User Profile for IBM DRDA over TCP/IP:

[https://www-912.ibm.com/s\\_dir/slkbase.NSF/643d2723f2907f0b8625661300765a2a/9e61c67ade70359986256afd007cc9f1?OpenDocument](https://www-912.ibm.com/s_dir/slkbase.NSF/643d2723f2907f0b8625661300765a2a/9e61c67ade70359986256afd007cc9f1?OpenDocument)

IBM AnyNet Configuration (SNA over TCP/IP) 1:

[https://www-912.ibm.com/s\\_dir/slkbase.nsf/1ac66549a21402188625680b0002037e/c5057870085200f8862565c2007cdc47?OpenDocument](https://www-912.ibm.com/s_dir/slkbase.nsf/1ac66549a21402188625680b0002037e/c5057870085200f8862565c2007cdc47?OpenDocument)

IBM AnyNet Configuration (SNA over TCP/IP) 2:

[https://www-912.ibm.com/s\\_dir/slkbase.nsf/1ac66549a21402188625680b0002037e/coe073485d35f074862569da0063626f?OpenDocument](https://www-912.ibm.com/s_dir/slkbase.nsf/1ac66549a21402188625680b0002037e/coe073485d35f074862569da0063626f?OpenDocument)

### **IBM Info Center Documentation:**

Source System Security in a TCP/IP Network — Server Authentication - V5R3:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/ddm/rbae5sourcesecurity.htm>

Source System Security in a TCP/IP Network — Server Authentication - V5R4:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/ddm/rbae5sourcesecurity.htm>

**Part 1 of this article series:**

Data Queue APIs and CL Commands, Part 1:

<http://www.systeminetwork.com/article.cfm?id=53542>

**This article demonstrates the following data queue API:**

Retrieve Data Queue Description (QMHQRDQD) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qmhqrdqd.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/53685\\_149\\_DataQueue2.zip](http://www.pentontech.com/IBMContent/Documents/article/53685_149_DataQueue2.zip)

**Source URL:** <http://iprodeveloper.com/rpg-programming/apis-example-data-queue-apis-and-cl-commands-part-2>