


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

APIs by Example: Physical File Triggers and the QDBRTVFD API

[iPro Developer](#)
[Carsten Flensburg](#)

Carsten Flensburg

Tue, 04/24/2012 - 6:00am

Accessing physical file trigger details is easy with this API

[Click here](#) to download the code bundle.

 To report code errors, email [iprodeveloper.com](mailto:carsten@iprodeveloper.com)

In the words of the API manual, the Retrieve Database File Description (QDBRTVFD) API "allows you to get complete and specific

information about a file on a local or remote system." This statement is certainly no exaggeration. At times, you may find the amount and complexity of the information returned by the QDBRTVFD API slightly overwhelming. To show how this API works, I exploit QDBRTVFD to gain access to details about the triggers associated with a specified physical file.

When it comes to creating and managing physical file triggers, we can use native CL commands and SQL statements, but so far, we still lack a CL command for listing and working with the triggers defined for a physical file. Employing the QDBRTVFD API and User Interface Manager programming techniques, however, provides an easy solution for that insufficient situation. Here, we look at the new Work with Physical File (WRKPFTRG) command, based on the trigger information that the QDBRTVFD API makes available. (To download the code bundle for this command, go to iprodeveloper.com/code.)

QDBRTVFD's Parameters

Let's begin with a look at the QDBRTVFD API parameter list (Figure 1), as documented by IBM in the online Information Center. QDBRTVFD will put the requested file information in the first parameter, *Receiver variable*. The second parameter tells the API how much space is available in terms of receiver variable length in bytes. If QDBRTVFD deems the initial storage allocated for the return variable as insufficient, the API returns the actual size needed in the *Bytes available* subfield of the return variable; this amount of storage is then reallocated and the API call subsequently repeated.

Figure 1: The QDBRTVFD API required parameter group

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Qualified returned file name	Output	Char(20)
4	Format name	Input	Char(8)
5	Qualified file name	Input	Char(20)
6	Record format name	Input	Char(10)
7	Override processing	Input	Char(1)
8	System	Input	Char(10)
9	Format type	Input	Char(10)
0	Error Code	I/O	Char(*)

`Default Public Authority: *USE`

The third API parameter, *Qualified returned file name*, is an API output parameter used to communicate the qualified name of the file for which information is returned. You'll find this parameter useful if you employ one of the special values *LIBL or *CURLIB to specify the library part of the fifth API parameter, *Qualified file name*. Here, you submit the qualified name of the file for which to retrieve the requested file information. In this case, the actual library name resolved is returned as part of the third parameter.

The fourth API parameter—*Format name*—identifies the type of file information to return. The types of file information available and their associated return format names are as follows:

- File definition template—return format FILDo100
- Format definition template—return format FILDo200
- Key field information template—return format FILDo300
- Trigger information template—return format FILDo400

Each format name identifies an often very complex hierarchy of structures. In each format's header structure, you'll find offsets to the relevant associated substructures applicable to the requested format name; these substructures in turn provide offsets to deeper nested substructures. In your program, you then jump from structure to structure to reach the information you want to access.

This procedure can present a challenge because of the complex and layered structure of the return formats. For more details about this parameter, please refer to the online API documentation listed in Find Out More, below. Here, you'll also find links to previous APIs by Example articles discussing QDBRTVFD, the techniques involved in calling this API, and processing the produced output.

The sixth API parameter, *Record format name*, applies only to return format FILDo200. In this parameter, you indicate the name of the record format in the specified file used to generate the file description. Special value *FIRST, meaning the first record format found, is supported for this parameter.

Next, the *Override processing* parameter defines whether the API should honor current file overrides in effect for the job calling the API. The eighth parameter—*System*—communicates on which system the specified file should reside, either the local or a remote system, as indicated by two of the special values available for this parameter, *LCL and *RMT, respectively. A third value, *FILETYPE, lets you request information about files on both the local and remote systems, depending on the file type.

The *Format type* parameter, used only with format FILDo200, controls whether the logical formats returned are internal or external. The parameter value *EXT points to external formats, whereas parameter value *INT points to internal formats. Because you're probably already familiar with the final API parameter (the standard API error data structure), I'll leave it out of scope for now.

Given the requirement of retrieving the triggers defined for a specified physical file, this article takes advantage of the FILDo400 API return format. Compared with other return formats, FILDo400 is quite simple. The header format—Qdb_Qdbftrg_Head—defines not only general trigger information but also the offset to the Qdb_Qdbftrg_Def_Head structure, which is actually an array containing one entry for each of the triggers defined for the file.

The following approach provides the foundation for retrieving trigger information:

1. The Qdb_Qdbftrg_Head header structure is based on the space pointer pQdb_Qdbftrg_Head, which is assigned the address of the API return variable.
2. The Qdb_Qdbftrg_Def_Head array structure is based on the space pointer pQdb_Qdbftrg_Def_Head.
3. To get from the Qdb_Qdbftrg_Head header structure to the first entry of the Qdb_Qdbftrg_Def_Head array structure, you add the offset defined by the Qdb_Qdbftrg_Head structure's subfield Off_Ent_Num1 to the pQdb_Qdbftrg_Head pointer and store the result in the pQdb_Qdbftrg_Def_Head pointer.
4. To get to the next array entry, simply add the Qdb_Qdbftrg_Def_Head entry length defined by the structure's Def_Len subfield to the pQdb_Qdbftrg_Def_Head pointer. Repeat this procedure only for the number of Qdb_Qdbftrg_Def_Head entries available, as defined by the Qdb_Qdbftrg_Head header structure's subfield Num_Trgrs, because it's otherwise unpredictable what storage the pQdb_Qdbftrg_Def_Head pointer references.

Following the Process

To see how this recipe transforms to RPG/IV, check out the code in the CBX246 command processing program's LodTrgInf subroutine. Running the CBX246 program in the source debugger lets you follow the process as it unfolds. (For a list of the downloadable code files, see "How to Compile," below.)

When prompted, the WRKPFTRG command appears as Figure 2 shows.

Figure 2: Work with Physical File Triggers (WRKPFTRG) command prompt

```

- - - - -
- - - - -

                                Work with Phys File Triggers (WRKPFTRG)

Type choices, press Enter.

Physical file . . . . . Name
Library . . . . . *LIBL Name, *LIBL, *CURLIB
Output . . . . . * * , *PRINT

- - - - -
- - - - -

```

You'll also find help text provided for the command and its parameters. Running WRKPFTRG for file CUS001F in library PRODLIB on my system produced the list panel in Figure 3, displaying the two triggers defined for the mentioned file.

Figure 3: Work with Physical File Triggers list panel

```

-----
-----

                                Work with Physical File Triggers
WYNDHAMW

                                11-02-12
16:41:05
File . . . . . :    CUS001F          Trigger count . . :    2

Library . . . . :    PRODLIB

Type options, press Enter.

    2=Change    3=Copy    4=Remove    5=Display    6=Print    8=Work with
object

                                Trg

    Opt  Program      Library      Nbr  State      Oper  Type  Time
Event
      CRM022I        PRODLIB        1  *ENABLED  *YES  *SYS  *BEFORE
*INSERT
      CRM022I        PRODLIB        2  *ENABLED  *YES  *SYS  *BEFORE
*UPDATE

Bottom
Parameters or command

===>

F3=Exit      F5=Refresh    F6=Add physical file trigger    F8=Work
with file

```

F11=View 2	F12=Cancel	F22=Display entire name	F24=More
keys			
- - - - -	- - - - -	- - - - -	- - - - -
- - - - -			

In addition to the trigger information listed in Figure 3, other views offer extra details, including trigger name and trigger library as well as other operative attributes associated with the trigger, such as *Allow repeated change*, *Update condition*, and more. The list options let you change, copy, remove, display, and print the selected trigger(s) as well as work with the trigger program object. You can find further documentation on the list panel, columns, list options, and function keys in the help text panel group included with the WRKPFTRG command.

Find Out More

- [Retrieve Database File Description \(QDBRTVFD\) API](#)
- ["APIs by Example: Analyzing Logical Files Using the QDBRTVFD File API"](#)
- ["APIs by Example: Working with Database Files, Fields and More"](#)
- ["APIs by Example: Displaying and Locating a Physical File's Access Paths"](#)
- ["APIs by Example: Print File Field Description"](#)

How to Compile

Below you'll find instructions on how to create the Work with Physical File Triggers command. This article includes the following sources:

- CBX246—RPGLE: Work with Physical File Triggers
- CBX246E—RPGLE: Work with Physical File Triggers - UIM Exit Pgm
- CBX246H—PNLGRP: Work with Physical File Triggers - Help
- CBX246P—PNLGRP: Work with Physical File Triggers - Panel Group
- CBX246V—RPGLE: Work with Physical File Triggers - VCP
- CBX246X—CMD: Work with Physical File Triggers
- CBX246M—CLP: Work with Physical File Triggers - Build command

To create all above command objects, compile and run the CBX246M CL program, following the instructions in the source header. You'll also find compilation instructions in the respective source headers of the individual sources.

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-physical-file-triggers-and-qdbrtvfd-api>