

APIs by Example: Working with Database Files--the API Way!

[Carsten Flensburg](#)

Sat, 03/29/2014 - 1:55pm



A Swiss Army knife provides a variety of useful tools all in one place. In this installment of APIs by Example, I'll present a similar toolset that helps database programmers get the job done efficiently: the *Work with Database Files* (WRKDBF2) command. The WRKDBF2 command is the result of me performing repetitive tasks against all sorts of database files whenever I was involved in research, design, or programming efforts. I eventually came to the conclusion that a single command offering shortcuts to all the database tools I used most often would be a great time saver; thus, the WRKDBF2 command was born.

WRKDBF2 Command Basics

The WRKDBF2 command is a simple list panel utility. I employ the *Open List of Objects* (QGYOLOBJ) API to retrieve all the file objects that meet the file criteria specified as the command's primary parameter. For all physical files, the Retrieve Database File (QDBRTVFD) API is called to ensure that only database files are selected. The WRKDBF2 command's list panel offers access to a variety of CL commands that specifically target database files. (A number of these commands have been presented in earlier installments of the APIs by Example series; I've included links in the Find Out More section below.) The *Open List of Objects* (QGYOLOBJ) API parameter list documented in Figure 1 involves up to 15 parameters.

Figure 1: Open List of Objects (QGYOLOBJ) API parameters:

Open List of Objects (QGYOLOBJ) API

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	List Information	Output	Char(80)
4	Number of records to return	Input	Binary(4)
5	Sort information	Input	Char(*)
6	Object and library name	Input	Char(20)
7	Object type	Input	Char(10)
8	Authority control	Input	Char(*)
9	Selection control	Input	Char(*)
10	Number of keyed fields to return	Input	Binary(4)
11	Key of fields to return	Input	Array(*) of Binary(4)
12	Error Code	I/O	Char(*)

Optional Parameter Group 1:

13	Job identification information	Input	Char(*)
14	Format of job identification information	Input	Char(8)

Optional Parameter Group 2:

15	Auxiliary storage pool (ASP) control	Input	Char(*)
----	--------------------------------------	-------	---------

Twelve of those are required, and the remaining three are split into two optional groups. I'll briefly discuss each of the parameters as I walk you through the parameter list.

The Parameters

The first parameter is where the QGYOLOBJ API returns the requested object information. The data returned is divided into two parts: a fixed portion that identifies the object and a variable format that reflects the object attributes requested in the key field array defined as the API's 11th parameter. The fixed and variable parts together form a return record, and depending on the size of the parameter variable, many records might be

returned in one API call.

Each available object attribute is defined by a numeric key, and adding a key value to the key field array asks the API to return the associated object attribute in the variable part of the receiver variable parameter. This method lets you scope the API return information to only the details that you specifically require, in contrast with fixed-format data structures, which are often defined as a set of attributes that are mutually related. This implies that if you need one attribute, you'll also get all the others, irrespective of the performance penalty incurred by the API when it retrieves information for which you have no immediate use.

The second API parameter defines the length of the receiver variable and tells the API how much storage it can safely address when returning the requested information. The third parameter is a standard data structure that allows Open List APIs to pass information about the list returned back to the API caller (e.g., total number of records, actual number of records returned, length of each record, list and completion status). This information is valuable in terms of navigating the list as well as assessing whether the requested information is accurate and complete.

The fourth parameter is a single integer instructing the API how to build the list and how many records to return. Basically, an Open List API builds the list either synchronously or asynchronously. The former method means the entire list is built before the API call completes; this approach is requested by submitting the special value -1. Any value greater than -1 will cause a secondary server job to be submitted, and this server job will then be in charge of building the list. In this case, the initial API call completes as soon as the number of records specified have been retrieved.

The next parameter gives the API caller the option to define the sort order that will be applied when the return list is built (this is another open list concept). Technically, the mechanism is implemented using the same convention and method as the Sort (QLGSORT) and the Sort Input/Output (QLGSRTIO) APIs. You can ignore this option, but it lets you easily control and adapt the output order of the returned list, so it can come in handy.

The sixth parameter defines the object and the library name that identifies the objects to include in the list. Special values are supported for both parameter elements. Proper research of this parameter is amply rewarded because it supports a number of very useful list configuration options. For details, check out the API documentation. You can further qualify the object selection with the seventh parameter, which lets you specify the object type as either a specific object type or the special value *ALL.

The eighth and ninth parameters work in conjunction and let you narrow down the returned object list based on an authority control format and a selection control format, respectively. The two control formats allow you to, for example, select objects based on the object authority assigned to the current user, ignore adopted authority, or select damaged objects only. Again, the API documentation has all the details—for now, just the note that in certain circumstances, these two parameters can be very useful.

The 10th parameter is a single integer containing the number of key field values submitted in the key field array parameter (the 11th parameter, which was explained earlier). This information lets the API know exactly how many key field values to process. The 12th parameter is the standard API error data structure, which you use to determine whether the QGYOLOBJ API call completed successfully. To check for a successful completion, make sure the *Bytes available* subfield in the error data structure is equal to zero. If no error occurred, it's safe to proceed.

Optional parameter group one consists of parameters 13 and 14. The 13th parameter, a *Job identification information* data structure, is used to identify the job, or thread within a job, for which objects can be searched using the thread's library list. This means you can use the QGYOLOBJ API to list objects found in another job's QTEMP library, for example, or use another job's library list to locate the objects being listed. If you specify this parameter, you must also specify a library name—either QTEMP or one of the special values (*CURLIB, *LIBL, or *USRLIBL).

Parameter 14 identifies the data structure format used in the 13th parameter to identify the job or thread used to select the object list. You use format JIDFo000 to identify the current job and JIDFo100 and JIDFo200 to identify another job. Format JIDFo200 is used if you have access to a thread handle to the job in question. Thread handles, which are returned by some Work Management APIs, are the simplest way to identify a thread in a job. If you don't have a thread handle, you can use format JIDFo100, where the job or thread is identified by a specified thread identifier.

Optional parameter group two comprises only parameter 15, the *Auxiliary storage pool (ASP) control format*. This format is used to tell the ASP to search when the object list is built, and it's quite a simple data structure (refer to the API documentation for the details). To learn more about the programming techniques involved in using the Open List APIs, read my article "[APIs at Work – with Jobs.](#)"

Master the QGYOLOBJ API

Despite the many parameters, once you've used the QGYOLOBJ API a few times it's fairly easy to adapt and tailor your code to meet new requirements. The WRKDBF2 command is based on the requested *Object type* *FILE object list of being returned by the QGYOLOBJ API. The selection is further qualified because the *Object attribute* is specified as either a physical file (PF) or a logical file (LF). As I mentioned earlier, physical file objects are validated to exclude source physical files from the presented list (logical files are, by definition, database files).

The WRKDBF2 command prompt and list panel are shown in Figure 2 and Figure 3, respectively.

Figure 2: Work with Database Files (WRKDBF2) command prompt:

```
Work with Database Files (WRKDBF2)

Type choices, press Enter.

File . . . . . Name, generic*, *ALL
Library . . . . . *LIBL Name, *LIBL, *CURLIB...
File type . . . . . *ALL *ALL, *PF, *LF, *DDMF
Sort order . . . . . *FILE *FILE, *LIB
```

Figure 3: Work with Database Files list panel

```
Work with Database Files                                WYNDHAMW                                16-03-14  11:05:28

Type options, press Enter.
 1=PDM   2=Change  3=Copy   4=Delete  5=Display data  6=Display fields
 7=Access paths  8=File desc  9=Run query  10=WRKDBF  11=Analyze LF
12=Triggers 13=Change desc 14=Clear  15=Un-delete 16=Generate SQL

Opt  File      Library  Type  Text
----  ---      -
QADBCCST  QSYS      PF    Constraint Field Usage Information
QADBCKCL  QSYS      LF    LF for QADBCCST by Constraint Library and
QADBCLNK  QSYS      LF
QADBFCST  QSYS      PF    File Level Constraint Cross Reference File
QADBFDEP  QSYS      PF    Cross reference dependency file
QADBIATR  QSYS      LF    Cross reference logical file by attribute
QADBIFLD  QSYS      PF    Cross reference physical file
QADBILFI  QSYS      LF    Cross reference logical file by long name
QADBILLB  QSYS      LF    Cross reference logical file by long libra
QADBIREL  QSYS      LF

More...

Parameters or command
===>
F3=Exit      F4=Prompt    F5=Refresh   F9=Retrieve   F11=Display full text
F12=Cancel   F13=Start SQL session  F14=Run SQL statements  F24=More keys
```

All in all, 16 list options are available in the *Work with Database Files* list panel, giving you the option of investigating or managing various aspects of the listed database files with the associated database CL commands:

```
1=PDM           (WRKMBRPDM)
2=Change        (CHGPF/CHGLF)
3=Copy          (CPYF)
4=Delete        (DLTF)
5=Display data  (DSPPFM)
6=Display fields (DSPPFD2)
7=Access paths  (DSPPFAP)
8=File desc     (DSPFD)
9=Run query     (RUNQRY)
10=WRKDBF/UPDDTA (WRKDBF/UPDDTA)
11=Analyze LF   (ANZLFITG)
12=Triggers     (WRKPFTRG)
```

```

13=Change desc      (CHGOBJD)
14=Clear            (CLRPFM)
15=Un-delete       (UNDEL2)
16=Generate SQL DDL (GENSQLDDL)

```

The actual CL command hiding behind the option's textual description is shown in parentheses. Note that for option 10, the WRKDBF command will be listed if you have the WRKDBF product installed on your system; otherwise, the native Update Data (UPDDTA) command will be shown. The GENSQLDDL command has been updated to reflect new capabilities that have been added to the QSQGNDDL API.

Option 15 is the un-delete command UNDEL2 written by Dave McKenzie, and it's included with the [code download](#) for this article. UNDEL2 version 2.0.7 is (according to Dave) expected to be the final version of this excellent and very useful command, so be sure to grab it while you can. (Note that the UNDEL2 command is also part of Bill Reger's WRKDBF utility.) Thanks to Dave for this lifesaving utility!

How to Compile

Below you'll find instructions on how to create the Work with Data Base Files (WRKDBF2) command and all its associated commands and objects. The following sources are included with the code download available with this article:

CBX123 – RPGLE – Print File Field Description - CPP

CBX123H -- PNLGRP – Print File Field Description - Help

CBX123X – CMD -- Print File Field Description

CBX123M – CLP – Print File Field Description - Build Command

CBX176 – RPGLE – Generate SQL Data Definition Statements - CPP

CBX176H -- PNLGRP – Generate SQL Data Definition Statements - Help

CBX176X – CMD -- Generate SQL Data Definition Statements

CBX176M – CLP – Generate SQL Data Definition Statements - Build command

CBX177 – RPGLE – Display Physical File Access Paths - CPP

CBX177E -- RPGLE – Display Physical File Access Paths - UIM Exit Program

CBX177H -- PNLGRP – Display Physical File Access Paths - Help

CBX177P -- PNLGRP -- Display Physical File Access Paths - Panel group

CBX177V – RPGLE – Display Physical File Access Paths - VCP

CBX177X – CMD -- Display Physical File Access Paths

CBX177M – CLP – Display Physical File Access Paths - Build command

CBX178 – RPGLE – Display File Field Description 2 - CPP

CBX178E -- RPGLE – Display File Field Description 2 - UIM Exit Program

CBX178H -- PNLGRP – Display File Field Description 2 - Help

CBX178P -- PNLGRP -- Display File Field Description 2 - Panel Group

CBX178X – CMD -- Display File Field Description 2

CBX178M – CLP -- Display File Field Description 2 - Build Command

CBX215 – RPGLE -- Analyze Logical File Integrity - CPP

CBX215H – PNLGRP – Analyze Logical File Integrity - Help

CBX215P – PNLGRP -- Analyze Logical File Integrity - Panel Group

CBX215V -- RPGLE -- Analyze Logical File Integrity - VCP

CBX215X -- CMD -- Analyze Logical File Integrity

CBX215M -- CLP -- Analyze Logical File Integrity - Build command

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

CBX267 – RPGLE -- Work with Database Files - CCP

CBX267E – RPGLE -- Work with Database Files - UIM Exit Program

CBX267H – PNLGRP -- Work with Database Files - Help

CBX267P -- PNLGRP – Work with Database Files - Panel Group

CBX267V -- RPGLE -- Work with Database Files - VCP

CBX267X -- CMD -- Work with Database Files

CBX267M – CLP -- Work with Database Files - Build Command

To create the above commands and associated objects, compile and run the CBX267M CL program, which will call all the other CL programs included in the above list, following the instructions in the source header. You'll also find compilation instructions in the respective source headers of the individual sources.

Find Out More

[Work with Database File](#) (Bill Reger)

[“APIs at Work — with Jobs”](#)

[“APIs by Example: Analyzing Logical Files Using the QDBRTVFD File API”](#)

[“APIs by Example: Displaying and Locating a Physical File's Access Paths”](#)

[“APIs by Example: Working with Database Files, Fields, and More”](#)

["APIs by Example: Reverse Engineering Database Files and Objects to SQL DDL Statements"](#)

["APIs by Example: Physical File Triggers and the QDBRTVFD API"](#)

["Teach Your Old DB2 New Tricks"](#)

["Simplify DDS to SQL Conversion"](#)

["Use SQL CREATE OR REPLACE to Improve DB2 for i Object Management"](#)

["Replacing a DDS Physical File with an SQL Table"](#)

[Open List of Objects \(QGYOLOBJ\) API](#)

[Retrieve Database File Description \(QDBRTVFD\) API](#)

[Sort \(QLGSORT\) API](#)

[Sort Input/Output \(QLGSRTIO\) API](#)

Content Classification: Influencer

Source URL: <http://iprodeveloper.com/application-development/apis-example-working-database-files-api-way>