

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

APIs by Example: Move and Rename Object API, and IBM CL Command Reuse

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 01/27/2011 (All day)

In the previous installment of the APIs by Example column (see ["APIs by Example: Command Definition API and API XML Output Processing,"](#) December 9, 2010, article ID 65648), I demonstrated the Retrieve Command Definition (QCDRCMDD) API and how its output can be parsed with RPG's XML-SAX opcode. Today, I continue with that theme, this time from the perspective of how you can use existing IBM CL commands to speed up and ease the task of creating your own CL commands.

To help me do so, I've created a CL command front end to the Rename Object (QLIRNMO) API, in turn simplifying the task of moving and replacing an object, moving and renaming, or both, all in one single operation. I've also included a brief CL program giving an example of how to take advantage of the Move and Rename Object (MOVRNMOBJ) CL command in the context of retaining a number of previous versions of a transaction work file created and processed, on, for example, a daily basis.

The QLIRNMO API is more than just a combination of the functions provided by the Move Object (MOV OBJ) and the Rename Object (RNM OBJ) commands. It also provides an option to replace the target object when it already exists, a situation in which both of the aforementioned commands would fail. This means that your code does not have to handle this situation separately but instead can set the QLIRNMO API's Replace object parameter in accordance with the behavior you'd want for that particular situation.

In this example, I'm specifying the QLIRNMO API directly as the MOVRNMOBJ command's CPP. All QLIRNMO API parameters can be mapped directly to the parameter types offered by a command definition. In this case, I decided not to create a separate CPP to call the QLIRNMO API. If I had chosen to include a command completion message or wanted to offer special values for the MOVRNMOBJ command parameters, I would need a CPP to convert them into valid API parameters. If I did that, I'd have to spend some time wrapping up the API call, but in this case, I saved myself the extra work.

The only challenge with such an approach is the standard API error data structure that most APIs require. A command definition, however, can specify a command parameter as a constant. This means that you specify the value of the parameter directly in the command definition statement for that parameter. The command will not display constant parameters on a command prompt but instead will simply pass the specified parameter value to the CPP. This allows me to exploit the standard API error-handling convention, dictating that passing a four-byte integer with the value zero as the error parameter to the API will cause the API to communicate error conditions by returning an exception message straight to the API caller, in this case being the caller of the command.

The MOVRNMOBJ command's error parameter as defined in the command definition source:

Parm	ERROR	*Int4	+
	Constant(0)		

Another command definition facility that I chose to include in this example is a command parameter choice program (CP). In previous articles, I have presented examples of choice programs that come in handy when you want to control the text appearing in the command prompt to the right of a parameter's input field prompt, the choice text, as well as the list of valid parameter input values presented when you press function key F4 from within a parameter input field prompt. Normally, the command will take both sets of information from the values specified for the command parameter's Single values, Special values, Values, Range, etc., attributes. Using a CP lets you override this behavior. I discuss CPs in a little more detail in the article titled "APIs by Example: Journal APIs Solving Spooled Files Mysteries," to which I've provided a link at the end of this article.

For the task at hand, I use a CP to provide the list of object types available to the MOVRNMOBJ command's Object type (OBJTYPE) parameter. To do so, I refer to the QLIRNMO API's documentation, which explains that the object types supported by the API reflect those supported by the MOV OBJ and RNMOBJ commands, respectively. Taking advantage of some of the code I developed last time, allowing me to retrieve command definition statements in the form of XML-formatted data using the Retrieve Command Definition (QCDRCMDD) API, and extracting the command information from the API output using RPG/IV XML parsing function XML-SAX, makes it quite trivial to retrieve the list of valid object types for the MOV OBJ and RNMOBJ commands. Because the CP is run at time when the user has specified neither the object to process nor the target object name and library, the list will have to be a combination of both lists of object types.

The CP therefore retrieves the list of valid object types for both commands and consolidates the returned lists into one single list of object types. Once created, the CP (named CBX222C) is specified on the MOVRNMOBJ command's OBJTYPE parameter using the CHOICE(*PGM) CHOICEPGM() command definition attributes:

Parm	OBJTYPE	*Char	10	+
	Min(1)			+
	Expr(*YES)			+
	Choice(*PGM)			+
	ChoicePgm(CBX222C)			+
	Prompt('Object type')			

However, the OBJTYPE parameter definition does not enforce the list of values returned by the CP. To actually restrict the user from specifying an object type not included in the list, I employ a command validity checking program (VCP). The VCP can verify that the user selected an option from the CP's list. Because a VCP is not run until after the user has entered the command parameters, I use it to check which operation will be done. If the user entered a target object library different from the library of the object to process, a move operation is being performed; otherwise, a rename operation is assumed. I then retrieve the object type list for the applicable command and verify that the entered object type is valid for the attempted operation.

Because both the CP and the VCP need to retrieve the command parameter's list of supported values, I've written a program that returns the special values supported for a particular parameter of a particular command. Another option would be adding a prototype and procedure interface to the code, and creating a service program instead of a program. Either option works and ensures that you're up to speed immediately, given the future need of such functionality. For details on the

process of retrieving command definition statements in XML format, as well as converting and parsing this information, please look up the December 2010 APIs by Example article and code, which I mentioned and linked to at the beginning of this article, and to which you'll also find a link at the end of this article.

At this point, all that remains is to put the pieces together and create the MOVRNMOBJ CL command. Once the MOVRNMOBJ command has been successfully created and you prompt the command, you'll see the following display panel:

Move and Rename Object (MOVRNMOBJ)		
Type choices, press Enter.		
Object		Name
Library	*LIBL	Name, *LIBL,
*CURLIB		
Object type		*ALRTBL, *AUTL,
*BNDDIR...		
To object		Name
Library		Name
Replace object	*NO	*NO, *YES, *PGM
From ASP device	*	Name, *,
*CURASPGRP, *SYSBAS		
To ASP device	*ASPDEV	Name, *,
*ASPDEV...		

In essence, the MOVRNMOBJ command's parameter list, including the hidden ERROR parameter, simply reflects the QLIRNMO API's corresponding parameter list. You specify the object to rename, the object type, the new object name and library, and the target object replace option. To help demonstrate the MOVRNMOBJ command in action, I've included a CL test program with the code accompanying this article. Please follow the instructions provided below to create and call the test program. The test program will create a test file by the name and in the library specified by the test program's two input parameters, here's an example:

```
CALL PGM(CBX222T) PARM('CBX222F' 'QGPL')
```

The above call command will create the CBX222F transaction work file in library QGPL and archive that file in a backup cycle retaining the last seven work files processed. The archived files are named CBX222Fo1, CBX222Fo2, . . . , CBX222Fo7. Each time the CBX222T test program is called, all archived files are renamed using the MOVRNMOBJ command. CBX222Fo7 is replaced by CBX222Fo6, CBX222Fo5 is renamed CBX222Fo6, and so forth, and the newly "processed" CBX222F is archived as CBX222Fo1.

You can run the CBX222T CL program in the source debugger to see for yourself how the events unfold. You must run the CBX222T program seven times to create a full cycle of archived files. As you will note, using the MOVRNMOBJ command specifying REPLACE(*YES) allows the program to ignore whether the target object of the rename operation already exists. If it does, it is simply replaced, and if it does not, no problem. Check out the command help text for all the details concerning the MOVRNMOBJ command in general and the REPLACE parameter in particular.

A final piece of reuse of IBM CL commands and components relates to the aforementioned help text panel group associated with the MOVRNMOBJ command. Because the QLIRNMO API documentation refers to the restrictions applying to the RNMOBJ and MOVOBJ command, as far as supported object types are concerned, I decided to include hyperlinks to the global command help text for both these commands in the help text for the MOVRNMOBJ command.

In the restrictions part of the global command help text for MOVRNMOBJ command, the following section appears:

- o All restrictions applying to the `_Move Object (MOVOBJ)` command, as well as the `_Rename Object (RNMOBJ)` command also apply to this command. Please refer to the respective command's help text for all the details.

Placing the cursor on either the `_Move Object (MOVOBJ)` or `_Rename Object (RNMOBJ)` part of the above help text and pressing Enter will cause the respective command's global help text module to be displayed. In case you're unfamiliar with this technique, here's how you go about it. Use the Display Command (DSPCMD) command to display the command information for the command whose help text you want to include in your own help text panel group. On the resultant display's page two, make a note of the *Help panel group* name as well as the *Help identifier*. The latter is typically the command name:

Display Command Information			
Command	: RNMOBJ	Library	:
QSYS			
Message file for prompt text	:	QCPFPMPT	
Library	:	QDEVELOP	
Message file	:	QCPFMSG	
Library	:	*LIBL	
Current library	:	*NOCHG	
Product library	:	*NOCHG	
Bookshelf	:	*LIST	

```

Help panel group . . . . . : QHLICMD

Library . . . . . : *LIBL

Help identifier . . . . . : RNMOBJ

Prompt override program . . . . . : *NONE

Enabled for graphical user interface . : *YES

Threadsafe . . . . . : *COND

Coded character set ID . . . . . : 37

```

Immediately following the :PNLGRP. tag in your help text panel group source you specify the panel group name, as in the example below.

```

:PNLGRP.
:IMPORT PNLGRP='QHLICMD' NAME='*'.

```

The NAME='*' tag following the panel group name defines that all help modules not found in this panel group or explicitly specified on other :IMPORT tags are to be located in the panel group specified on this :IMPORT tag. You can have only one :IMPORT tag in a panel group specifying a "catch all" NAME='*' tag. All other imported panel group modules must be named explicitly using the NAME tag, as in the example below:

```

:IMPORT PNLGRP='QHSYCMD' NAME='RTVAUTLE/ALL'.
:IMPORT PNLGRP='QHSYCMD' NAME='RTVAUTLE/CHANGE'.
:IMPORT PNLGRP='QHSYCMD' NAME='RTVAUTLE/USE'.

```

At the point in your panel group help text source where you want to place the hyperlink, you then specify a :LINK PERFORM='DSPHELP...' statement pointing to the help identifier located earlier for the command in question. You specify the help identifier only because it's the global help section and not one of the command's parameters, as in the following example taken from today's code:

```

All restrictions applying to the
:LINK PERFORM='DSPHELP MOVOBJ'.
Move Object (MOVOBJ)
:ELINK.
command, as well as the
:LINK PERFORM='DSPHELP RNMOBJ'.
Rename Object (RNMOBJ)
:ELINK.
command also apply to this command. Please refer to the respective
command's help text for all the details.

```

The DSPHELP UIM dialog command displays the specified UIM help module, as in the above example in which a link is established to both the RNMOBJ and MOVOBJ global command help. The

panel group containing the help module to display must either be specified on the DSPHELP dialog command or, as discussed and employed in this example, on an :IMPORT tag in the panel group header section.

You can also import help text for each individual command parameter. In today's example, I've taken the help text for the OBJTYPE parameter from the corresponding RNMOBJ command parameter. This type of import follows the convention applying to the hyperlink reference above, and since I already declared the RNMOBJ command's help text panel group on the :IMPORT tag specifying all modules on the NAME='*' tag, all other RNMOBJ help modules are readily available. So I simply specify the following :IMHELP tag to embed the OBJTYPE help text in my panel group:

```
:HELP NAME='MOVRNMOBJ/OBJTYPE'.
Object type (OBJTYPE) - Help
:IMHELP NAME='RNMOBJ/OBJTYPE'.
:EHELP.
```

Had you wanted to create a hyperlink to the command parameter instead, which sometimes can be very practical too, you'd have specified the help identifier qualified name of the parameter in question. I've included an example below:

```
See the help text for the
:LINK PERFORM='DSPHELP RNMOBJ/OBJTYPE'.
OBJTYPE
:ELINK.
parameter.
```

For more information and examples of including help text with your own CL command, please check out the links pointing to earlier articles covering this topic at the end of this article.

This APIs by Example includes the following sources:

```
CBX222C  -- RPGLE  -- Move and Rename Object - Choice Program
CBX222H  -- PNLGRP -- Move and Rename Object - Help
CBX222V  -- RPGLE  -- Move and Rename Object - VCP
CBX222X  -- CMD    -- Move and Rename Object

CBX222L  -- RPGLE  -- Retrieve Command Parameter Value List

CBX222T  -- CLP    -- Move and Rename Object - Test
CBX222M  -- CLP    -- Move and Rename Object - Build commands
```

To create all these objects, compile and run the CBX222M program following the instructions in the source header. You'll also find compilation instructions in the respective source headers. As for the test program CBX222T, you'll need to run the Create CL Program command specified in the program source header and subsequently call the program specifying two parameters. A 10-byte parameter specifying the name of the transaction work file to archive, and another 10-byte parameter specifying the name of the library containing the work file. The program will create the work file for the purpose of conducting the test, if the file does not already exist.

IBM Technical Document:

[Defining Choice Exit Programs for Parameter Prompts in CL Commands](#)

Related articles:

[APIs by Example: Command Definition API and API XML Output Processing](#)

[APIs by Example: Journal APIs Solving Spooled Files Mysteries](#)

[APIs by Example: Do You Need Help?](#)

[Put Help on Your CL Commands: It's Easy!](#)

This article demonstrates the following APIs:

[Rename Object \(QLIRNMO\) API](#)

[Retrieve Command Definition \(QCDRCMDD\) API](#)

[iconv\(\) function](#)

[QtqIconvOpen\(\) function](#)

[iconv_close\(\) function](#)

[Retrieve the source code for this API example.](#)

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-move-and-rename-object-api-and-ibm-cl-command-reuse>