


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

APIs by Example: Command Definition API and API XML Output Processing

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 12/09/2010 (All day)

The Retrieve Command Definition (QCDRCMDD) API is useful if you need to retrieve information from a command object about the CL command definition statements required to create the command object. Knowing that the API exists and what it's capable of, however, is only half the way when it comes to taking advantage of the information returned by the QCDRCMDD API. The QCDRCMDD API returns information in the form of either a return variable or a stream file. If you settle for the return variable, your first challenge is to perform the conversion required to interpret the UTF-8 Unicode character set used by the API to create the output information.

Opting for the stream file will help you overcome this issue, because the IFS stream file APIs required to open and read the output file can perform character set conversion implicitly. You will, however, need to write the code employing these APIs to make this approach work. Irrespective of the output destination chosen, the next challenge is to parse the XML format used to structure and organize the command definition information. Today's issue of APIs by Example takes a closer look at the aforementioned programming challenges as well as introduces you to four new CL commands that make it easier to get the job done.

As for the XML part of the equation, I've naturally taken advantage of the XML parsing functions, the XML-INTO and XML-SAX opcodes, that have been added to RPG IV in recent releases. As luck would have it, a number of useful articles covering this topic in much detail have been published, also in the *System iNetwork Programming Tips* newsletter. If you need to brush up on RPG IV's XML parsing capabilities, see the links to these articles at the end of this one.

The first thing I want to do is actually be able to create the XML command definition data. In order to do so, I have to build the Retrieve Command XML Source (RTVCMDXML) CL command, which is basically a front end to the QCDRCMDD API. Here's the RTVCMDXML command prompt:

Retrieve Command XML Source (RTVCMDXML)

Type choices, press Enter.

Command	Name
Library *LIBL	Name, *LIBL,
*CURLIB	

```

Output file . . . . .

Format . . . . . *FULL *BASIC, *FULL

```

The Retrieve Command XML Source (RTVCMDXML) command retrieves information from a CL command (*CMD) object and generates XML source statements that describe the command. The generated command information XML source is called Command Definition Markup Language or CDML. The CDML source is stored in the stream file specified in UTF-8. UTF-8 (CCSID 1208) is a Unicode format that resembles ASCII but allows the data to be stored compactly and shared easily between IBM i systems and any other system that supports the UTF-8 format.

The CDML elements and attributes closely resemble the command definition statements used to create CL commands:

1. CMD (Command) statement
2. PARM (Parameter) statement
3. ELEM (Element) statement
4. QUAL (Qualifier) statement
5. DEP (Dependency) statement
6. PMTCTL (Prompt Control) statement

IBM in addition supplies a Document Type Definition (DTD) specification located in the IFS path /QIBM/XML/DTD/QcdCLCmd.dtd for the definition of the CDML tag language returned by the QCDRCMD API. Run the following command to display the content of the QcdCLCmd.dtd file:

```
DSPF STMF ('/QIBM/XML/DTD/QcdCLCmd.dtd')
```

As input to the RTVCMDXML command, you specify the qualified name of the command for which to retrieve the command definition information as well as a path identifying the stream file to receive the API output. The final command parameter defines the specification level of the returned command information. Format *BASIC returns CDML source that describes the CL command information needed to build a valid command string for the command, and format *FULL returns CDML source that describes all the command definition statements used to create the command. The latter is a superset of the information returned by the former.

To create a stream file named rtvobjd.xml in directory QOpenSys containing XML formatted command definition statements for the Display Object Description (DSPOBJD) command, you'd run the following command:

```
RTVCMDXML CMD(DSPOBJD) OUTFILE('/Qopensys/dspobjd.xml')
```

To inspect the resultant stream file, you can use your preferred XML editor. If you don't have such a tool readily available, you can download a copy of the freeware XML editor called *XML Marker*. I've included a Google query locating the download area for this tool at the end of this article. If you want to make the retrieved command information available in a program, you have a number of options

available. As far as native RPG IV options are concerned, you'd be looking at employing either the XML-SAX or XML-INTO opcode.

Irrespective of the option chosen, you'd need to work with XML data in order to identify the structure of the XML data as well as the XML elements and attributes and their respective values. This information is crucial whether you want to map the XML document to one or more data structures as supported by the XML-INTO opcode or you decide to capture the XML document content in an XML handler being registered for either the XML-INTO or XML-SAX opcode. As I mentioned, you'll find the details of both approaches discussed in the articles previously published, so I'll leave that part out of scope for now.

However, one of these articles, "RPG's XML-SAX Opcode," by Scott Klement, includes a couple of programs illustrating the XML events and XML content paths involved in processing arbitrary XML data. The program sources are named PrintXml.rpgle and ShowEvents.rpgle in the accompanying zip-file. I've been using Scott's programs whenever I need to quickly get an overview of or analyze XML documents in more detail. At one point, I therefore decided to add a command interface to those two programs in order to make it faster and even more convenient to establish the structure and element and attribute values of any given XML-document.

One of the commands is called List XML Document Events (LSTXMLEVT):

List XML Document Events (LSTXMLEVT)

Type choices, press Enter.

XML document

The List XML document events (LSTXMLEVT) command documents the events that occur as the specified XML document is being processed by the XML-SAX operation code in RPG IV. The produced list is printed with your job's spooled output. The other command is called List XML Document Content Path (LSTXMLCNTP):

List XML Document Content Path (LSTXMLCNTP)

Type choices, press Enter.

XML document

The LSTXMLCNTP command tracks the position and path of all XML elements in the specified XML document and prints this information together with the data values associated with each element. For more details on the output produced by both commands, refer to Scott's article, which thoroughly explains the basics of resolving an XML document into XML events as well as XML element and attribute paths.

Anyway, once you've completed the investigation of the XML document structure, you should be able to make an educated decision about the XML parsing options to choose and be armed with the information necessary to identify the element and attribute values to extract. As an example of how this knowledge could then be put into action, I've created the List Command Parameter Info (LSTCMDPRMI) CL command:

```

List Command Parameter Info (LSTCMDPRMI)

Type choices, press Enter.

Command . . . . . Name

Library . . . . . *LIBL Name, *LIBL,
*CURLIB

```

The LSTCMDPRMI command produces a printed list containing information such as keyword, type, return value, length, default value, and prompt text about each parameter associated with the specified command name. Below, you'll find an excerpt of the list produced for the Create RPG Module (CRTRPGMOD) command:

```

12/06/10 16:18:08          System: NOVASTAR          Command
Parameter Information

Command name . . . . . : CRTRPGMOD

Library . . . . . : QSYS

Keyword      Type      Rtn value  Length  Min   Max   Dft value
Prompt text
CMDFLAG      CHAR                32       0
MODULE      QUAL      NO                0     1
Module
NAME                10       0     1     *CTLSPEC

```

	NAME		10	0	1	*CURLIB
Library						
SRCFILE	QUAL	NO		0	1	
Source file						
	NAME		10	0	1	QRPGLESRC
	NAME		10	0	1	*LIBL
Library						
SRCMBR	NAME	NO	10	0	1	*MODULE
Source member						
SRCSTMF	PNAME	NO	5000	0	1	
Source stream file						
GENLVL	INT2	NO		0	1	10
Generation severity level						
TEXT	CHAR	NO	50	0	1	*SRCMBRTXT
Text 'description'						
OPTION	INT2	NO		0	20	
Compiler options						
DBGVIEW	INT2	NO		0	1	*STMT
Debugging views						
OUTPUT	CHAR	NO	1	0	1	*PRINT
Output						
OPTIMIZE	INT2	NO		0	1	*NONE
Optimization level						

Looking at the LSTCMDPRMI command processing program CBX2213 reveals that, following the QCDCRMDD API call and subsequent conversion of the returned command information, the parsing of the XML data is performed in the statement below:

```

/Free

    Xml-Sax  %Handler( XmlHandler: ComVar )  %Xml( XmlStr:
'doc=string' );

/End-Free

```

The statement in essence declares the XML data to be processed to be located in the XmlStr variable and that the subprocedure to be called for each XML parsing event is named XmlHandler. Again, the mechanics and interface of an XML handler are documented and explained in the XML-SAX and XML-INTO articles I mentioned earlier, but basically the code in the XmlHandler subprocedure will be responsible for identifying and retrieving the XML elements and attributes being passed to it. If you want to see how to code for this and verify how it works, I suggest you use your favorite source debugger against the CBX2213 CPP and add break points in the XmlHandler subprocedure to ensure that the execution of this is captured in the debugger during execution of the LSTCMDPRMI command.

The LSTCMDPRMI command uses the QCDCRMDD API return variable destination option and consequently handles the data conversion of the command XML information returned by the API from Unicode UTF-8 to the current job's Coded character set identifier (CCSID). This conversion is

performed by the `iconv()` Code Conversion APIs. Initially I had planned to use Convert a Graphic Character String (QTQCVRT) API for this purpose because this single API call approach is a bit simpler to implement as opposed to the three API calls required by the `iconv()` method.

It quickly turned out, however, that the XML-formatted API output exceeded the string length constraint of 32767 bytes of the QTQCVRT API. For smaller strings and minor volume data conversion requirements, the QTQCVRT API is a convenient alternative to the slightly more complex `iconv()` functions, so I've included a version of the LSTCMDPRMI CPP using the QTQCVRT API to let me demonstrate the somewhat special Feedback (FB) error-handling process employed by this API.

The QTQCVRT API reports any error condition in an array of three 32-bit two's complement binary values (12 bytes). The status code is a non-negative number in the first 16 bits, and the reason code is a non-negative number in the second 16 bits. The specific meanings of the status code and associated reason code values are documented in a table included at the end of the QTQCVRT API documentation. The CPFA33F message ID in message file QCPFMSG can be used to format and return error information to the API caller, as the example I've included with this article demonstrates.

To see how it works, rename the CBX2213 program to CBX22131, rename the CBX22132 program to CBX2213, and run the following command:

```
LSTCMDPRMI CMD(CHGLINSDLC)
```

If everything goes according to my expectations, you should receive the following exception message in return:

```

Additional Message Information

Message ID . . . . . : CPFA33F      Severity . . . . . :
40
Message type . . . . . : Information

Date sent . . . . . : 06-12-10      Time sent . . . . . :
20:07:42

Message . . . . . : Error occurred during data conversion.

Cause . . . . . : An error occurred using the CDRCVRT API to
convert data
between the coded character set identifier (CCSID) for the job
1208 and the
display device 277. The feedback code is X'00080005'. See the
Character
Data Representation Architecture Level 2 Reference, SC09-1390, to
determine
the cause of the error.
```

Recovery . . . : Correct the error and try the request again.

Once you've completed the test, remember to rename program CBX22131 back to CBX2213 in order for the LSTCMDPRMI to work correctly also for commands producing comprehensive amounts of XML data; at least up to and including 65200 bytes of XML data. I don't know if that has any practical implication, but beyond that limit further code changes will need to be implemented in order to avoid an exception during the command definition XML data processing.

This APIs by Example includes the following sources:

```
CBX221  -- RPGLE  -- Retrieve Command XML Source - CPP
CBX221H -- PNLGRP -- Retrieve Command XML Source - Help
CBX221X -- CMD    -- Retrieve Command XML Source

CBX2211 -- RPGLE  -- List XML Document Events - CPP
CBX2211H -- PNLGRP -- List XML Document Events - Help
CBX2211V -- RPGLE  -- List XML Document Commands - VCP
CBX2211X -- CMD    -- List XML Document Events

CBX2212 -- RPGLE  -- List XML Document Content Path - CPP
CBX2212H -- PNLGRP -- List XML Document Content Path - Help
CBX2212X -- CMD    -- List XML Document Content Path

CBX2213 -- RPGLE  -- List Command Parameter Information - CPP
CBX2213H -- PNLGRP -- List Command Parameter Information - Help
CBX2213X -- CMD    -- List Command Parameter Information
CBX22132 -- RPGLE  -- List Command Parameter Information - CPP (error
handling)

CBX221M -- CLP     -- Retrieve Command XML Source - Build commands
```

To create all these objects, compile and run the CBX221M program following the instructions in the source header. You'll also find compilation instructions in the respective source headers.

IBM RPG/IV XML documentation:

[Debug](#)

[XML Operations \(*\)](#)

[RPG and XML \(*\)](#)

(*) Use the Next Page button at page end to read through all pages

XML-Related articles:

[RPG's XML-SAX Opcode](#)

[Processing XML Docs with RPG—Simplified](#)

['Real World' Example of XML-INTO](#)

[PTFs for Version 6.1 Enhance RPG's XML-INTO](#)

[Convert Data Between CCSIDs](#)

[XML editor XML Marker](#)

This article demonstrates the following APIs:

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

[iconv\(\) function](#)

[QtqIconvOpen\(\) function](#)

[iconv_close\(\) function](#)

[Convert a Graphic Character String \(QTQCVRT\) API](#)

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

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-command-definition-api-and-api-xml-output-processing>