System iNetwork

- Subscribe
- My account
- Log out
- Contact Us
- Advertise

Search

GO

- Forums
- Archives
- Code
- Blogs
- Podcasts
- Webcasts
- e-Learning
- Guides
- Newsletters
- About Us

- Contact Us
- About the Network
- Tech Editor Profiles
- Editorial Calendar
- Writers Kit
- Advertise

- RPG Programming
- Other Languages
- Application Development
- Application Modernization
- Database/SQL
- Availability
- Security

R.P.G. XML

Article ID: 65249
Posted September 1st, 2010
in

- Database/SQL

Read, Prepare, Generate an XML document for any DB2 file
By:
David Andruchuk

Click here to download the code bundle.
To report code errors, email **SystemiNetwork.com**

In the January 2009 issue I demonstrated how to use an SQL stored procedure to generate an XML document from an IBM i file using the K.I.S.S. design principle, Keep It Short and Simple (see "K.I.S.S. XML," article ID 62565). But what about tried and true RPG, you ask? Is it possible to generate an XML document with IBM i's most popular of languages? The answer is yes. Using the R.P.G. design principle—Read, Prepare, and Generate—you can accomplish the same task for any table on the i system.

This article shows just how to do that using a command, panel group, CLLE programs, and an SQL RPG program to generate an XML document from an existing DB2 file. I chose to use a command/panel group/CLLE program front end for the SQL RPG because old school is still good school.

First, I begin by explaining just how simple it is to use RPG for the utility tasks often given to us at a moment's notice.

Front-End Processing

I have long been an advocate of using a command interface to gather the data required for a program's execution. A command—when used with a panel group for help instructions, a validity-checking program for command input validations, and a control language program to process the main program for execution—assures that the main program will execute with valid parameter values.

While I don't spend time on all the code contained in the command, panel group, and two CLLE programs because they are available for download on the *System iNEWS* code website, a few notes are in order to explain the workings of this program combination for those that have not used one or more parts of it.

Command. The command in Figure 1 accepts three parameters: file, document, and directory.

File indicates the DB2 file to generate the XML for, document names the generated XML document, and directory is where the IFS will store the generated XML stream file. Both the file and directory parameters are required for input, and each value specified must exist on the system for execution to continue.

Panel group. The panel group in Figure 2 is accessed by pressing the F1=Help key, which is cursor-position sensitive, anywhere on the command screen.

The panel group provides a user with specific application help for a command, display screen, menu, etc. Developers can create a panel group in the User Interface Manager (UIM). The development language is similar in style to HTML code. For more information on panel groups, reference the IBM "Application Display Programming" manual (SC41-5715-02).

CLLE validity-checking program. The validation-checking program validates the command-parameter data before initiating the program to process. If it finds an error, then the program displays the command screen, and the status message area indicates any error(s) (Figure 3), allowing the operator to correct the error before processing continues.

Error handling code in the CLLE validity-checking program (Figure 4) uses the IBM-provided messages CPD0006 and CPF0002.

Message CPD0006 is used by validity-checking programs to send an immediate message in the message data parameter. Any diagnostic message (*DIAG) also can be used for this if you choose to not use CPD0006.

After all diagnostic messages are sent, the validity-checking program must send message CPF0002 using the SNDPGMMSG command. When the system receives message CPF0002, it sends message CPF0001 to the calling program to indicate that errors have been found. More information on the use of a validity-checking program is available in the IBM "CL Programming V5R3" manual (SC41-5721-06).

Once all input has been validated, control passes to the program and execution continues.

SQL RPGLE Code

When used with embedded SQL, RPG provides powerful file processing with very little coding. The R.P.G. design principle requires a module that reads any file—simple or complex as the structure may be—passed to it and builds an XML document from the data and column names in the file.

Using the IBM table SYSCOLUMNS in the QSYS2 library, you can retrieve the column names for the supplied DB2 file and store them in a data structure array. The column names are then used to create an SQL select script that retrieves all the data contained in those columns and stores them in a data structure array.

The program example provided in the downloadable code has limits of 100 columns and 100 values per column. Additionally, each value can be at most 128 characters, so you will have to consider that if any table exceeds these limits.

Data Definition Specifications

A prototype and procedure interface accepts the passed parameters file, library, and XML document (Figure 5).

To store the column names and column data values retrieved from the DB2 file, I've created two data structure arrays: one named Columns and the other named Values (Figure 6). The data structure arrays let me store up to 100 column names (each of which is a maximum of 10 characters) and 100 column values (each of which contains up to 128 characters) from the DB2 file. Should a file have more than 100 columns or values greater than 128 characters, you will have to modify these parameters to fit your specific needs.

A third data structure array named Values_n is created with 100 null value fields to store any null value data retrieved from the DB2 file (Figure 7).

Calculation Specifications

After specifying the arrays, I create an SQL cursor to retrieve the column names for the DB2 file from the SYSCOLUMNS table in QSYS2. SYSCOLUMNS is an SQL view maintained by DB2 that contains all the field names and characteristics for all tables on your system. The cursor is opened and the column names fetched into the Columns data structure array (Figure 8). The work value NbrofColumns is set to the value returned in the SQL Communications Area data structure (SQLCA) to tell how many columns were returned before closing the cursor.

Next, I retrieve the DB2 file data for the column names retrieved from the SYSCOLUMNS table. To do so, I loop through the retrieved column names stored in the Columns data structure array and build an SQL SELECT statement for each of the column names. Each value retrieved for a column from the DB2 file is converted to a character string when it is stored in the Values data structure array so that I need not worry about the field data type when processing.

The cursor is opened and the column values are put into the Values data structure array (Figure 9). The work value NbrofValues is set to the value returned in the SQLCA to tell how many values were returned before closing the cursor.

Now I can begin to build the XML document from the columns and values we have stored from the two SQL fetches.

As was the case with the SQL stored procedure exercise, I build a temporary file in QTEMP to contain the generated XML document data. Once the file is created, I insert the XML declaration and root element data (Figure 10) because each XML document must contain the declaration and root elements.

Since I do not know what the column data values contain in terms of characters, I need to cleanse the data before I can generate the XML.

I check each byte of the value data's 128 characters retrieved from the Values data structure array for the existence of predefined XML character entities that will cause problems for the XML document. These character entities must be replaced with a predefined replacement text. That way, when these characters are processed in the XML document or an HTML page, the appropriate substitution occurs. (For more details about XML and HTML character entity references, see the Wikipedia page "List of XML and HTML character entity references") (Figure 11).

Once the data is cleansed, I can begin wrapping each of the data values stored in the Values data structure array with column names from the Columns data structure array formatted as XML data.

To identify a row of data retrieved from the file, I insert a child element at the beginning of each data row. The XML child element acts as a wrapper for all the columns and associated data I will be generating in the XML document (Figure 12).

Then, for each of the 128-byte data structure array values, I retrieve the column data value from the Values array and build an SQL statement that consists of the Columns array column name bracketed with the required XML brackets (XBL, XBR, or XBLC) and our Values array Value### data in character format. The format will read as column name value enclosed in the appropriate XML left (XBL) or right (XBR) bracket, the column data value, and then the column name value enclosed in the XML left close (XBLC) and right bracket. Once the SQL statement is formatted, I simply insert the row in the temporary XML table in QTEMP (Figure 13).

Once the data for each Values data structure array has been read, processed, and generated as XML data in the temporary XML table, I insert the closing child element end tag to identify each row of data retrieved from the file and close out the wrapper noted above (Figure 14).

Once all the data is complete and the XML document has been generated, I insert the ending root element tag to properly close out the XML document (Figure 15).

All that remains is to copy the generated XML data to a stream XML document in the IFS (Figure 16). To accomplish this task, I use the CPYTOIMPF command as an argument to the system command QCMDEXC procedure. I code a command variable with the necessary instructions and passed XML document directory and name. The QCMDEXC requires two parameters: the command with parameters to execute and length of the command passed to it.

And one final housekeeping task: drop the temporary work table created in QTEMP that held the XML document data (Figure 17).

Executing the Utility

From a command line I can prompt the RPGGENXML command and enter the required parameters to execute (Figure 18). Using the same Employee Master table as the "K.I.S.S. XML" article (Figure 19), as well as the same IFS folder /TEMPXMLDOC, I generate the same Employee.XML document (Figure 20) produced using the SQL stored procedure in "K.I.S.S. XML."

By using RPG and the power of embedded SQL, you can quickly reformat any provided DB2 table and create a self-described exchange document to share with an internal or external source. I'm hopeful that you now have the basic foundation to design and generate your own XML documents from DB2 tables.

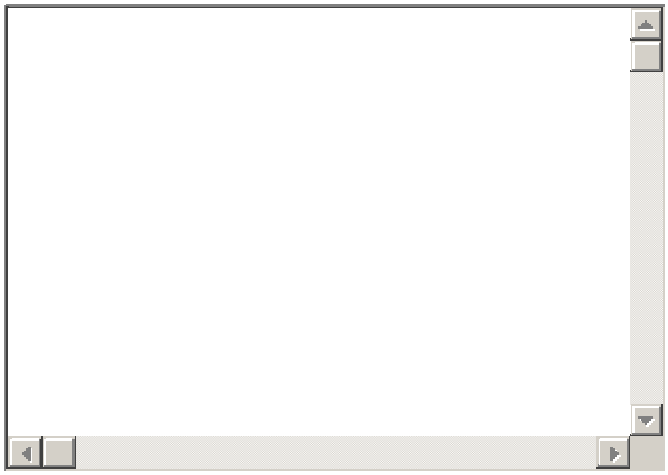
I would like to thank Doug Davie, a senior software engineer at SoftLanding Systems, for his help and guidance in completing this article.

David Andruchuk *is a senior system architect for Computer Systems Design Associates in Crystal Lake, Illinois. Starting his IT career on an IBM S/32, he has progressed through the midrange platform offerings during his 25-year career.*

Post new comment

Your name: pwesemann@sunga...

Comment: *



Input format

Filtered HTML

- Web page addresses and e-mail addresses turn into links automatically.
- Allowed HTML tags: <a>   <cite> <code>   <dl> <dt> <dd>
- Lines and paragraphs break automatically.

Full HTML

- Web page addresses and e-mail addresses turn into links automatically.
- You may use [block:*module=delta*] tags to display the contents of block *delta* for module *module*.

More information about formatting options

Math Question: * 11 + 7 =

Solve this simple math problem and enter the result. E.g. for 1+3, enter 4.

Preview comment

Acceptable Use Policy

Related Links

- Integrating XML with DB2 for i 7.1

- SQL your XML on IBM i
- Use a UDTF and SQL to Query XML Files
- Hibernate Your JDBC
- iSeries Treasure Hunt

ProVIP Sponsors

Featured Links

Sponsored Links