

# 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](#)

- **Systems Management**

- **Networking**

- **IT Mgmt/Careers**

- 

- 

- 

- 

[Home](#) » [Content](#)

## SQL your XML on IBM i

Article ID: 64663

Posted April 1st, 2010

in

- [Database/SQL](#)

Use common SQL processing to drill into your XML data

By:

[David Andruchuk](#)

You can use common SQL processing to drill into your XML data, and this article demonstrates how. This article takes off where two previous articles ended. Two years ago, co-author Kent Milligan of IBM Rochester and I showcased an enhancement that was new in IBM i 5.2. That enhancement gave DB2 for i (then called DB2 for i5/OS) support for user-defined table function (UDTF) capabilities. As we mentioned in that article, you can use UDTFs from SQL interfaces to access nonrelational data sources, such as S/36 files and stream files residing in the IFS.

In this article I again use the power of this unsung feature to decompose an XML document we created in the article "[K.I.S.S. XML](#)" (January 2009, ID 62565) and that is stored in an IFS folder. Using the RPG operation code XML-INTO (added to the RPG compiler in i 5.4) in an RPGLE program, you can transform the XML document to usable data, store it in a data structure array, and then return it as a result set to the SQL interface. That way you can use common SQL processing to drill into the XML data and view the results as if table data retrieved from disk. The result set returned to SQL is identical to the result set in the January 2007 article "[UDTFs: The Unsung DB2 Function](#)" (ID 20788) that was derived from S/36 flat table data.

First, a simple review is in order in case you missed the previous articles. DB2 for i shipped with support for user-defined table functions in i 5.2. The UDTF enhancement complemented

the user-defined scalar function (UDF) capability added way back in OS/400 V4R4. A user-defined function classified as scalar means that the function can only return a single value (or output parameter). That's where UDTFs come into play with their ability to return multiple values in the form of a table (or result set). I suggest you review the 2007 and 2009 articles so you are familiar with the code example as it has been some time since I touched upon these coding techniques.

## Coding the SQL Function

For this article I use the same code example provided with "UDTFs: The Unsung DB2 Function" in the current discussion on how to take advantage of the features afforded developers using UDTFs. You can download the code at [SystemiNetwork.com/code](http://SystemiNetwork.com/code) (January, 2007). For this exercise I simply renamed the Create Function and External Name values to SQLXMLUDTF from EMPDEPUDTF ([Figure 1](#)). I use the RUNSQLSTM command to create the SQL Function as I did in the previous article's exercise. You can download the code for this article at [SystemiNetwork.com/code](http://SystemiNetwork.com/code) (February 2010).

## Coding the RPGLE

As is the case for the SQL Function above, I use the same code example from "UDTFs: The Unsung DB2 Function" for the RPGLE program. Since for the most part the code remains unchanged in regards to the UDTF processing, this article concentrates on the code for the XML parsing that I added to the RPGLE. That code replaces the file I/O for the DEPMAST and EMPMAST tables because in this article I retrieve the data from an XML document residing in an IFS directory.

As is the case where you use existing code as a base for a current project, you should consider modernizing the code to make it more readable. For example, the CALLP and EVAL statements could have the operation code omitted and still function as expected. I chose not to modernize because I wanted to maintain the original coding as much as possible for clarity.

## Data Definition Specifications

The Prototypes and Procedure Interface code remains the same, so I simply change the names from EMPDEPUDTF to SQLXMLUDTF. These are used to communicate with the RPGLE program from the SQL interface.

I add two new Procedures, getEmplDept and ParseXML, to our data definition specifications:

```
d getEmplDept      PR
d ParseXML         PR
```

The getEmplDept procedure gets our decomposed XML data that is stored in a data structure array to return to SQL. The ParseXML procedure is used once at program initialization to decompose the XML document into the data structure.

To store our decomposed XML document data, a data structure array ([Figure 2](#)) is defined with the same names as the XML document Elements & Attributes names so that when the decompose process takes place, the XML Parser correctly populates the fields in our data structure array with the values found in the XML document.

I chose to store the XML directory and document name as a constant in a work field, but you could easily pass in the value as a parameter from the SQL interface as is done with the employee number:

```
d IFXMLDocument c const('/tempxml/doc/EmplDept.xml')
```

### Calculation Specifications

As was the case in the EMPDEPUDTF program, the main line of the calculations are minimal and handled by using procedure calls based on the CallType the SQL interface has passed.

```
select;
    when CallType = -1;
        callp $open();
    when CallType = *zero;
        callp $fetch();
    when CallType = 1;
        callp $close();
endsl;

return;
```

I modified the \$open procedure ([Figure 3](#)) to perform the XML decomposition with a call to the new ParseXML procedure because I need to parse the XML document just once and not at each call from the SQL interface when data is fetched.

I then modified the \$fetch procedure ([Figure 4](#)) to call the getEmplDept procedure for each element of the data structure array to get the data that was decomposed from the XML document replicating fetching data from disk as if stored in a database table.

The new procedure, ParseXML ([Figure 5](#)), is where I utilize the new built-in function added to the RPG ILE compiler in i 5.4. The XML-INTO built-in function facilitates decomposition of an XML document into data structures defined in the program using a non-validating parser.

The new operation codes and built-in functions make the decomposition of the XML data and population of data structure fields automatic and efficient without the developer having to manipulate storage to get the XML data into workable form.

The XML-INTO built-in function places into our Employees data structure array—specified as the parameter following the XML-INTO operation code—the values retrieved from our XML document starting at the Employees path when the XML data is decomposed using the parser. For more information on the XML-INTO built-in function and the parameters required, reference the IBM ILE RPG Language Reference manual (SC09-2508-06).

In [Figure 6](#), the getEMPMAS`T` procedure is replaced by the getEMPLDEPT to retrieve the data from the data structure array instead of accessing table data from disk as I did in the "UDTFs: The Unsung DB2 Function" article by looping through the 100 data structure array elements using the i1 index work field.

As was the case in the EMPDEPU`DTF` program, if the invoker of the UDTF supplies a valid employee identifier number, then the program searches the data structure array to return the personnel and department data for that specified employee. If no employee identifier number is specified, all the populated data structure array data is returned.

The setEndOfTable procedure is called to set the SQL\_STATE to '02000' and the returned parameters to NULL to indicate that no data was found, and control returns to the caller if either the passed employee identifier number is not found or the end of XML data is reached.

When a row in the data structure array is identified to be retrieved, the data for each of the data structure array fields is copied to the result output parameters (res\_empno, res\_firstname, etc). This processing is what lets DB2 pass the result data back to the invoker and make it appear like data from a newly created SQL table derived from our XML document.

That concludes the required changes to the RPGLE program from the "UDTFs: The Unsung DB2 Function" article, which demonstrated how to retrieve S/36 table data and return the results to the SQL interface. Showcasing the versatility of using the UDTF feature, you can see how the UDTF can be used for many special data needs, whether retrieving S/36 table data, parsing XML and returning as table data, or even using a specific business logic process unique to your business that you wish to be exposed as a returned table.

The final piece of this exercise is to demonstrate how to call the SQLXMLUDTF table function. The function, like all created UDFs and UDTFs, can be used from any SQL interface on the system, whether it be the interactive SQL interface (STRSQL) I am using in this article or embedded SQL, via a JDBC call, etc.

It should be made clear that the invoker can process the returned result set as if the data was retrieved from a DB2 table because that is the purpose of a UDTF.

The following SELECT statement will retrieve all the decomposed XML data stored in the data structure array and return it in the result set.

```
SELECT * FROM TABLE(sqlxmludtf('      ')) myudtf
```

The following SELECT statement will retrieve all the decomposed XML data stored for the employee with an identifier value of '000010'.

```
SELECT * FROM TABLE(sqlxmludtf('000010')) myudtf
```

While the passed employee identifier parameter was left in this exercise from the original "UDTFs: The Unsung DB2 Function" article, it is not required as the following would achieve the same result set being returned from the UDTF:

```
SELECT * FROM TABLE(sqlxmludtf('      ')) myudtf where empno = '000010'
```

## UDTF Coding Tips

The secondary threads used by DB2 for calling UDFTs can hold resources like locks that can hinder concurrent system activity on shared objects. Because of this, I recommend that user-defined table functions perform short, quick-running requests. By default, user-defined functions will timeout and end execution after 30 seconds. If you need to let a UDTF run for a longer period of time, the timeout value can be adjusted by specifying the UDT\_TIME\_OUT attribute in a QAQQINI query options file. On the topic of lock conflicts, it's also not a good idea to have the UDTF execute operations on the same database objects that are referenced by the SQL statement that invoked the UDTF in the first place.

The usage of threads by DB2 also makes it trickier to implement UDFs with the non-ILE languages, which are not thread safe. The non-ILE languages can be used for UDTFs, but it's highly recommended you use the ILE languages since they support threads. It was my intention to give you a deeper appreciation of the nonrelational data access and integrations problems that can be solved with UDTFs.

The simple examples that I reviewed in this article have given you the basic foundation to be able to design and code SQL, RPGLE, and external user-defined table functions for the purpose of accessing non-DB2 data with the SQL interface.

If your XML documents are more complex or large in size, the RPGLE code for the XML-INTO built-in function will have to be modified to use multiple data structures to simplify the XML decomposition and possible use of the %HANDLER parameter to decompose more than the 100 rows allowed for in the data structure array in the code example.

*David Andruchuk is a SR System Architect for Computer Systems Design Associates in Crystal Lake, Illinois. He began his IT career on an IBM S/32 and has progressed through the Midrange platform during his 25-year career.*

Bookmark/Search this post with:



[Printer-friendly version](#)

## 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> <em> <strong> <cite> <code> <ul> <ol> <li> <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: \* 1 + 0 =

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

Preview comment

[Acceptable Use Policy](#)

## Related Links

[Use a UDTF and SQL to Query XML Files](#)

[Hibernate Your JDBC](#)

[Podcast: What's New in DB2 on i?](#)

[Updated IFSDIR UDTF Provides Create Date](#)

[The Way to Character to Number, Number to Character on IBM i is SQL](#)

ProVIP Sponsors

- 
- 
- 
- 
- 

## Featured Links

## Sponsored Links

- Home
  - Subscribe Now
  - Advertise
  - Contact Us
  - Feedback
  - Terms of Use
  - Trademarks
  - Privacy Statement
- © 2010 Penton Media, Inc.