Incorporate web services into your SQL queries

Introducing HTTP functions on IBM DB2 for i

Yi Yuan
Nick Lawrence

January 23, 2014

In the past, resources supplied by IBM® DB2® for i have mainly included catalog objects, which users can query to discover details about the database. Recently, a new type of DB2 for i resource was added for database application developers: HTTP functions that are delivered as part of a recent DB2 for i Group PTF for IBM i 7.1. This article shows how to access web services using an SQL query with these functions and integrate the web service data with relational data. In this article, an example is used to demonstrate the usage of the HTTP functions and how to combine them with the DB2 for i built-in XML support.

Introduction

The Hypertext Transfer Protocol (HTTP) protocol is one of the most commonly used Internet protocols. Using HTTP, web services and other online information resources can be accessed through a Uniform Resource Locator (URL).

By making the new HTTP functions available as a system resource, DB2 for i has opened up a new avenue for database developers to incorporate web services using SQL.

The new HTTP user-defined functions (UDFs) and user-defined table functions (UDTFs) are written in Java™ and they exist in the SYSTOOLS schema. SYSTOOLS differs from other DB2 for i supplied schemas in that it is not part of the default system path. SYSTOOLS contains a set of DB2 for i supplied examples and tools. The tools and examples within SYSTOOLS are considered ready for use, but not part of any IBM product. So, they aren't subject to IBM service and support.

This article provides an overview of the new HTTP functions and an example to show how these functions can be used to request a web service and integrate the data result with the built-in XML support that is also part of DB2 for i 7.1 supports. The example shows how to design an SQL query that returns previously published entries from the DB2 for i blog.

Prerequisites

In order to use the HTTP functions with DB2 for i 7.1, the following software must be installed on the system.

- DB2 PTF Group SF99701 Level 23
- Java 1.6 or later (5761-JV1 Option 11, 12, 14, or 15)

HTTP functions overview

The HTTP UDFs and UDTFs are named using the following naming convention:

```
HTTP<method><data-type><verbose>
```

- **`method`** indicates the HTTP function. All the common HTTP methods are supported, including `POST`, `GET`, `PUT`, `DELETE` and `HEAD`. For more information about HTTP methods, you can refer to the Hypertext Transfer Protocol link in the Resources section.
- **`data-type`** can be `CLOB` or `BLOB` and it indicates the data type of returned HTTP response message or for some cases (such as `PUT` and `POST`), the type of the request message.
- DB2 for i provides both verbose and non-verbose versions of the HTTP functions for each method and data type combination.
  - Functions with 'verbose' suffix are table functions. The table functions return both: a response HTTP header and a response message in a result set. The response HTTP header includes a response code and header fields. The response code indicates whether the request was successful. The header fields contain additional information about the response.
  - Functions without 'verbose' are scalar functions and only the response message is returned.

For example, `HTTPPUTBLOBVERBOSE` is a table function that will use the `PUT` method to send and receive BLOB data, and `HTTPGETCLOB` is a scalar function that will use the `GET` method to retrieve a representation of a resource as a CLOB.

Besides the HTTP method functions, some helper functions are also provided to perform URL encoding and decoding and base64 encoding and decoding. The URL specification (RFC 1738) defines a set of special characters that need to be replaced with escape sequences (for example, if used in a query string of a URL). The helper functions `URLENCODE` and `URLDECODE` perform the URL encoding and decoding. Base64 encoding is commonly used to encode binary data to textual data on the web. The helper functions `BASE64ENCODE` and `BASE64DECODE` are provided for encoding and decoding base64 data.

Table 1 shows the signatures of the HTTP functions used in this article.

Table 1. Signature of HTTP functions used

Function name		
`httpGetBlobVerbose`	**Input parameter**	**Input parameter type**
	`URL`	`VARCHAR(2048)`
	`HTTPHEADER`	`CLOB(10K)`
	Output column	**Output column type**
	`RESPONSEMSG`	`BLOB(2G)`

	RESPONSEHTTPHEADER	CLOB(10K)
httpGetBlob	**Input parameter**	**Input parameter type**
	URL	VARCHAR(2048)
	HTTPHEADER	CLOB(10K)
	Return type	
	BLOB(2G)	
urlEncode	**Input parameter**	**Input parameter type**
	VALUE	VARCHAR(2048)
	ENCODING	VARCHAR(20)
	Return type	
	VARCHAR(4096)	

For the URLENCODE function, the VALUE parameter is the original string and the ENCODING parameter is used to specify encoding. When NULL is specified for the VALUE parameter, UTF-8 is used. UTF-8 is recommended by RFC 3986.

For a list of the HTTP functions that are not used in this example, refer to the white paper in the Resources section. You can find the source code of the HTTP functions under the integrated file system at:

/QIBM/ProdData/OS/SQLLIB/bin/systools_java_source.jar.

Retrieve DB2 for i blog entries

In this section, an example is provided to show how the HTTP functions are used to retrieve data from web service and how the data retrieved is processed in relational databases for better usage.

Figure 1 shows the result set we would like to retrieve from the query in the example which is to get the required entries from the DB2 for i blog. The result set contains published date, author, title, number of responses (comments), and the URL that identifies the blog post.

Figure 1. XMLTABLE result set

PUBLISHED	AUTHOR	TITLE	RES...	URL
2013-07-22 19:57:00.0...	Mike Cain	A Competitive Advantage...	0	http://db2fori.blogspot.com/2013/07/a-co...
2013-06-08 15:38:00.0...	Mike Cain	Advancing Your SQL Kno...	0	http://db2fori.blogspot.com/2013/06/adva...
2013-05-08 05:01:00.0...	Mike Cain	Expertise? Yes, I'll have so...	3	http://db2fori.blogspot.com/2013/05/expe...
2013-04-24 05:01:00.0...	Mike Cain	SEE	1	http://db2fori.blogspot.com/2013/04/see.h...
2013-04-09 17:58:00.0...	Mike Cain	Sustained Business Value -...	0	http://db2fori.blogspot.com/2013/04/susta...
2013-03-30 00:06:00.0...	Mike Cain	More Advice! Is this Goo...	0	http://db2fori.blogspot.com/2013/03/mor...
2013-03-25 15:57:00.0...	Mike Cain	As I See it: Victor Rozek is ...	0	http://db2fori.blogspot.com/2013/03/as-i-...
2013-03-13 00:27:00.0...	Mike Cain	Living Large	0	http://db2fori.blogspot.com/2013/03/livin...
2013-02-26 20:36:00.0...	Mike Cain	Visit the Home of Coke	0	http://db2fori.blogspot.com/2013/02/visit-...
2013-02-15 23:31:00.0...	Mike Cain	Integrating XML - Past, Pr...	0	http://db2fori.blogspot.com/2013/02/integ...
2013-02-06 22:31:00.0...	Mike Cain	What to Make of IBM i 7.1...	1	http://db2fori.blogspot.com/2013/02/what...

The first step in this example is to consult the application programming interface (API) documentation for the web service that needs to be accessed. You can refer to the documentation for the DB2 for i blog.The documentation tells us that the HTTP GET method can be used to return a list of all posts after a specific time, and the URL for this should look as shown in Example 1.

Example 1: URL format

```
http://db2fori.blogspot.com/feeds/posts/default?published-min=rfc3339_timestamp
```

In Example 1, `rfc3339_timestamp` is a timestamp in the format that is described by the RFC 3339 standard. An example of the RFC 3339 timestamp looks like "2013-05-12T23:20:50.52Z".

In order to construct a timestamp of the correct format, the UDF as shown in Example 2 is used. Using a UDF makes the code that constructs the URL a little easier to read.

Example 2. Function to create an rfc3339 timestamp

```
CREATE FUNCTION rfc3339_ts_format(in_time TIMESTAMP)
RETURNS VARCHAR(26)
LANGUAGE SQL
SET OPTION DATFMT=*ISO
 RETURN CONCAT( CONCAT(CAST(DATE(in_time) AS CHAR(10)),
                       'T' ),
                CHAR(TIME(in_time), JIS));
```

The URL can now be constructed with the SQL expression shown in Example 3. When specifying parameter data in a URL, the best practice is to make use of the `SYSTOOLS.URLENCODE` scalar function. The function recognizes special characters and adds escape characters where needed.

Example 3. Constructing the URL

```
CONCAT(CONCAT('http://db2fori.blogspot.com/feeds/posts/default?', 'published-min='),
       SYSTOOLS.URLENCODE(rfc3339_ts_format(CURRENT_TIMESTAMP - 6 MONTHS),'UTF-8'))
```

The next step is to set up the request headers. XML built-in functions provided by DB2 for i make it easy to process XML data within SQL. The blog web service supports the retrieval of data using an Atom format, which is based on XML. Therefore, the request header is coded so that the response is in the Atom format, as shown in Example 4.

Example 4. Request header for atom

```
<httpHeader>
    <header name="Accept" value="application/atom+xml"/>
</httpHeader>
```

At this point, it is possible to retrieve the XML response message using the `HTTPGETBLOBVERBOSE` table function. Example 5 shows the query statement.

Example 5. Query to retrieve a feed with blog posts

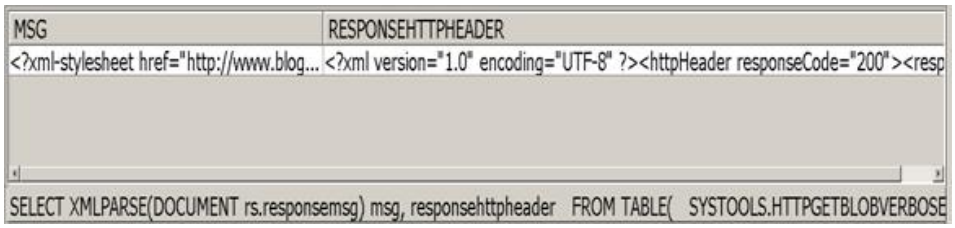
```
SELECT
   XMLPARSE(DOCUMENT rs.responsemsg) msg,
   rs.responsehttpheader
 FROM TABLE(
  SYSTOOLS.HTTPGETBLOBVERBOSE(
    -- URL --
CONCAT(CONCAT('http://db2fori.blogspot.com/feeds/posts/default?','published-min=' ),
    SYSTOOLS.URLENCODE(
       rfc3339_ts_format(CURRENT_TIMESTAMP - 6 MONTHS),
       'UTF-8'
    )),

    -- Header ---
    '<httpHeader>
     <header name="Accept" value="application/atom+xml"/>
    </httpHeader>')
) rs;
```

There are a few important decisions that were made in Example 5.

- The response message is returned from the table function as a `BLOB`. Using a `BLOB` data type for serialized XML data instead of a `CLOB` is usually the best practice, because it avoids issues where the character set of the data does not match the encoding declaration in the XML document.
- In the column list of the SELECT statement, `BLOB` is transformed into an XML value using the `XMLPARSE` function. The XML data type can be used with SQL and XML functions such as `XMLTABLE`, which is demonstrated later in this example.
- The verbose table function is used in this query, rather than a scalar function. When initially testing a web service, it is often easier to debug problems if the response HTTP header is available to examine.

The result set from the query in Example 5 is shown in Figure 2. The `RESPONSEHTTPHEADER` column contains an HTTP response code of `"200"`, which indicates that the request was successful.

Figure 2. Result set



The simplified content of the returned XML result of the MSG column looks as shown in Example 6.

Example 6. MSG result set column detail

```
<?xml-stylesheet href="http://www.blogger.com/styles/atom.css"
type="text/css"?>
<feed xmlns="http://www.w3.org/2005/Atom"
    xmlns:openSearch=http://a9.com/-/spec/opensearchrss/1.0/
    xmlns:blogger="http://schemas.google.com/blogger/2008"
```

```
    xmlns:georss="http://www.georss.org/georss"
    xmlns:gd="http://schemas.google.com/g/2005"
    xmlns:thr="http://purl.org/syndication/thread/1.0">
   <entry>
      <published>2013-07-22T14:57:00.000-05:00</published>
      <title type="text">A Competitive Advantage? It's all about the data.
      </title>
      <link rel="alternate" type="text/html"
           href="http://db2fori.blogspot.com/2013/07/a-competitive-advantage-its-all-about.html"
           title="A Competitive Advantage? It's all about the data."/>
      <author>
        <name>Mike Cain</name>
        <uri>http://www.blogger.com/profile/01481223716996299215</uri>
        <email>noreply@blogger.com</email>
      </author>
      <thr:total>0</thr:total>
   </entry>
   <entry>…</entry>
   ……
</feed>
```

The simplified content of the returned XML result of the `RESPONSEHTTPHEADER` column looks as shown in Example 7.

Example 7. RESPONSEHTTPHEADER result set column detail

```
<?xmlversion="1.0"encoding="UTF-8"?>
<httpHeaderresponseCode="200">
    <responseMessage>OK</responseMessage>
    <header name="GData-Version" value="1.0"/>
    <header name="HTTP_RESPONSE_CODE" value="HTTP/1.1 200 OK"/>
    <header name="Transfer-Encoding" value="chunked"/>
    <header name="Vary" value="Accept, X-GData-Authorization, GData-Version"/>
    <header name="Date" value="Thu, 01 Aug 2013 09:19:15 GMT"/>
    <header name="X-XSS-Protection" value="1; mode=block"/>
    <header name="Expires" value="Thu, 01 Aug 2013 09:19:15 GMT"/>
    <header name="Last-Modified" value="Mon, 22 Jul 2013 19:57:19 GMT"/>
    <header name="Content-Type" value="application/atom+xml; charset=UTF-8"/>
    <header name="Server" value="GSE"/>
    <header name="X-Content-Type-Options" value="nosniff"/>
    <header name="Cache-Control" value="private, max-age=0, must-revalidate, no-transform"/>
</httpHeader>
```

The XML document in Example 6 is not very helpful; what we need is to be able to shred or decompose the XML document into a relational database table, as shown in Figure 1, for better reference.

The structure of the XML document in the response message is explained by the documentation provided by the web service. An alternative approach for determining this information is to examine the document shown in the MSG column of Figure 2. After a sample document has been examined, the XPath expressions that locate the interesting parts of the XML document can be constructed.

The `XMLTABLE` function can be used to create the required result set, as shown in Example 8. For simplicity, the scalar `HTTPGETBLOB` function is used in Figure 8 instead of the `HTTPGETBLOBVERBOSE` table function that was used in Example 5. It would be possible to write this same query as a join between the `HTTPGETBLOBVERBOSE` table function and the `XMLTABLE` function. However, this example does not use the HTTP response headers, and therefore, the scalar function is sufficient.

Example 8. Using the XMLTABLE function to convert XML to a relational result set

```
SELECT published, author, title, responses, url
FROM
XMLTABLE(
  --------------- Namespace declarations -------------
  XMLNAMESPACES(
      DEFAULT 'http://www.w3.org/2005/Atom',
      'http://purl.org/syndication/thread/1.0' AS "thr"
  ),
  --------------- Row expression --------------------
  'feed/entry'
  --------------- Initial context -------------------
  PASSING
    XMLPARSE(DOCUMENT
      SYSTOOLS.HTTPGETBLOB(
       -- URL --
       CONCAT( CONCAT('http://db2fori.blogspot.com/',
          'feeds/posts/default?published-min='),
         SYSTOOLS.URLENCODE(
        RFC3339_TS_FORMAT(CURRENT_TIMESTAMP - 6 MONTHS),
        'UTF-8'
       )),

       -- header --
       '<httpHeader>
        <header name="Accept"
                   value="application/atom+xml"/>
        </httpHeader>'
        )
    )
  --------------- Result Set Columns ----------------
  COLUMNS
    published TIMESTAMP
        PATH 'published',
    author VARCHAR(15) CCSID 1208
        PATH 'author/name',
    title VARCHAR(100) CCSID 1208
        PATH 'link[@rel="alternate" and
                   @type="text/html"]/@title',
    responses INTEGER
        PATH 'thr:total',
    url       VARCHAR(4096) CCSID 1208
        PATH 'link[@rel="alternate" and
                   @type="text/html"]/@href'
    ) RS
ORDER BY PUBLISHED DESC
```

The `XMLTABLE` table function has several important components.

The `XMLNAMESPACES` declaration is used to define the namespaces that will be used in the XPath expressions. This example makes the default namespace, "http://www.w3.org/2005/Atom" for all of the unqualified elements used in the XPath expressions, and binds the prefix "thr" to the namespace "http://purl.org/syndication/thread/1.0".

The row expression indicates that the result set will contain one row for every 'feed/entry' element.

The `PASSING` clause defines the initial context that is used to evaluate the row expression. In this example, the XPath expression that is used in the row expression is relative to the root of the XML document that is returned from the `XMLPARSE` function.

The `HTTPGETBLOB` scalar function is used within the `PASSING` clause's expression to retrieve the response message. The parameters for the `HTTPGETBLOB` scalar function match with what was previously discussed in Figure 2. The response message is parsed into an instance of the XML data type (using the `XMLPARSE` function), and provided as a parameter of the `XMLTABLE` function.

The `COLUMNS` clause defines the columns that result from the `XMLTABLE` function. Each column has a name, an SQL data type, and an XPath expression. The item from the XML document that is identified by the XPath expression is converted to the SQL data type, and assigned to the column of the result set. The path expression is relative to the entry element that is being used to produce the row.

The result of the query in Example 8 matches with the result set that was shown in Figure 1.

Conclusion

The DB2 for i HTTP functions provide an easy way to access resources on the web from within an SQL statement. When combined with the DB2 for i built-in XML support, they provide the ability to incorporate web services to database directly and seamlessly. This article provides an example showing the basic usage of the HTTP functions to retrieve data from the web service and how to decompose the XML data to relational data.

Resources

- White paper: Accessing web services using DB2 for i HTTP UDFs and UDTFs
- Accessing RESTful services from DB2: Introducing the REST user-defined functions for DB2. (This article talks about the support for these functions on DB2 for z/OS and DB2 for Linux, UNIX and Windows)
- Hypertext Transfer Protocol (RFC 2616)
- URL specification (RFC 1738)
- Atom (standard)
- XMLTABLE tutorial
- Getting started with the XML Data Type Using DB2 for IBM i
- Using XML with DB2 for IBM i
- Now Introducing XML in SQL on DB2 for IBM i!
- DB2 for IBM i technology updates
- DB2 for i forum