Send and receive user-defined SOAP and REST messages from RPG

Integrated web service client APIs offer an easy approach to send payloads

By Nadir Amra, IBM Software Engineer
Published 01 June 2016

Abstract: The integrated web services client for Integrated Language Environment (ILE) has been used for years to send SOAP messages by generating stubs that hide the details about the SOAP messaging protocol. However, it did not allow users to send user-defined payloads over the Hypertext Transfer Protocol (HTTP) transport. This article discusses the enhancements in the integrated web services client that allow you to bypass the stubs and send user-defined requests.

Introduction

The integrated web services client for ILE has been used for years to send SOAP messages by generating stubs that hide the details about the SOAP messaging protocol. Users would set the fields in the stub-generated structures, invoke the web service operation, and receive the response. And this has worked great.

But what if you want to bypass the stubs and just send a user-defined payload, such as an Extensible Markup Language (XML) document that is not generated by the stubs? Or maybe you want to send a JavaScript Object Notation (JSON) payload as a Representational State Transfer (REST) request, for which stub generation currently does not provide support. What to do then?

The program temporary fixes (PTFs) that are needed for each of the supported releases of the IBM i operating system to use the APIs:

7.3: SI60805, SI60808
7.2: SI60806, SI60809
7.1: SI60807, SI60810

The answer to the questions is the integrated web services client. The client library has been enhanced with new application program interfaces (APIs) that enable users to send user-defined payloads. This article provides information on the APIs and some examples of API usages written in ILE RPG.

About the new transport APIs

The transport APIs may be used by client applications that want to control what is sent to a server and what is received from the server.

For RPG programming, the APIs and constants are defined in the *include* file: `/QIBM/ProdData/OS/WebServices/V1/client/include/Axis.rpgleinc`.

The following table summarizes the transport APIs. You can find more details about the transport API functions in the *Integrated Web Services Client for ILE programming* guide on the integrated web services website (see Resources).

Table 1. Transport APIs

Function	Is used to
axiscTransportCreate()	Create a transport object.
axiscTransportDestroy()	Destroy a transport object.
axiscTransportReset()	Reset the transport object to its initial state.
axiscTransportSetProperty()	Set a transport property.
axiscTransportGetProperty()	Get a transport property.
axiscTransportSend()	Send bytes over transport.
axiscTransportFlush()	Flush the transport of any buffered data.
axiscTransportReceive()	Receive data from the transport.
axiscTransportGetLastErrorCode()	Get transport error code from the last unsuccessful transport operation.
axiscTransportGetLastError()	Get transport error string from the last unsuccessful transport operation.

The typical flow of events when using the transport APIs is as follows:

1. Use the `axiscTransportCreate()` function to create a transport object. The URL to the web service is specified in the call to the function.
2. Set any transport properties (for example, connect timeout, HTTP method, HTTP headers, whether payload needs to be converted to UTF-8, and so on) using the `axiscTransportSetProperty()` function.
3. Send data (if any) using the `axiscTransportSend()` function. Data is buffered until the `axiscTransportFlush()` function is called. The data is automatically converted to UTF-8 unless the AXISC_PROPERTY_CONVERT_PAYLOAD transport property is set to "`false`" (in which case the data is sent as is).
4. Send the request to the client by invoking the `axiscTransportFlush()` function.
5. Receive the response to the request by calling the `axiscTransportReceive()` function. This API must be called even if no data is returned in order to consume the HTTP response to the request, which includes the HTTP response headers and status code. The data is automatically converted from UTF-8 to the job coded character set identifier (CCSID) unless the AXISC_PROPERTY_CONVERT_PAYLOAD transport property is set to "`false`" (in which case the data is returned as is).
6. Destroy the transport object by calling the `axiscTransportDestroy()` function.

Let us now look at a sample client that uses the APIs to perform a REST request.

About the REST APIs used in the article

The REST APIs that the client example (in this article) can invoke is based on the APIs documented in the article, Building a REST service with integrated web services server for IBM i, Part 3. The REST APIs developed in the article assumes a database of student registrations and allows you to retrieve, add, delete, and update student registrations using normal REST conventions. Table 2 shows a summary of the APIs that will be used in the sample code.

Table 2. REST API information for student registration example

REMOVE	**URL**	/*context-root*/students/*{id}*
	Method	DELETE
	Request body	None
	Returns	204 No content
		404 Not found
		500 Server error
CREATE	**URL**	/*context-root*/students
	Method	POST
	Request body	JSON
	Returns	201 Created
		409 Conflict
		500 Server error

GETALL	**URL**	/*context-root*/students
	Method	GET
	Request body	None
	Returns	200 OK and JSON
		500 server error

The student registration database contains the records shown in Figure 1.

Figure 1. Student registration database records

"studentID"	"firstName"	"lastName"	"gender"
823M934LA	Nadir	Amra	Male
826M660CF	John	Doe	Male
747F023ZX	Jane	Amra	Female

Using the APIs – Sending REST requests

The client application invokes the REST APIs as follows:

1. Removes a student registration using the DELETE HTTP method.
2. Creates a student registration using the POST HTTP method.
3. Retrieves all student registrations using the GET HTTP method.

The source code for the client is available in the Code Listings section at the end of this article. So, without further ado, let us go over the code as it uses the new client APIs. Figure 1 shows the beginning of the code.

Figure 1. Client application code (part 1 of 7)

```
**free                                                    1
Ctl-Opt DFTNAME(RESTCLIENT);

/COPY /QIBM/ProdData/OS/WebServices/V1/client/include/Axis.rpgleinc    2

DCL-S rc          INT(10);
DCL-S tHandle     POINTER;

DCL-S uri         CHAR(200);
DCL-S response    CHAR(32768);
DCL-S request     CHAR(32768);
DCL-S propBuf     CHAR(100);
```

Looking at Figure 1, you can find that the client code is using the ILE RPG compiler support for free-form code from column 1 to the end of line. This is indicated by specifying `**FREE` in column 1 of the first line (**1**). The other thing to notice is the `/COPY` statement (**2**) that is used to include the various client API function prototypes and related constants.

Free-form RPG PTFs:

7.2: SI58137
7.1: SI58136

The code in Figure 2 is the start of the logic that initiates a delete action of a student registration record.

Figure 2. Client application code (part 2 of 7)

```
// ------------------------------------------------------------------
//  Web service logic. The code will attempt to invoke a Web service.
// ------------------------------------------------------------------

// Uncomment to enable trace
// axiscAxisStartTrace('/tmp/axistransport.log': *NULL); 3

// Set URI in order to delete student
uri = 'http://localhost:10035/web/services/students/823M934LA'; 4

// Create HTTP transport handle.
tHandle = axiscTransportCreate(uri:AXISC_PROTOCOL_HTTP11); 5
if (tHandle = *NULL);
  PRINT ('TransportCreate() failed');
  return;
endif;

// Delete student registration, ID is part of URI
PRINT ('==Deleting student registration record');

propBuf = 'DELETE' + X'00'; 6
axiscTransportSetProperty(tHandle: AXISC_PROPERTY_HTTP_METHOD: %addr(propBuf));
flushAndReceiveData(); 7
```

In Figure 2, you can uncomment the line (**3**) that contains the function call to enable trace. If you do uncomment, the trace file will be created in file `/tmp/axistransport.log`. Recall from Table 2 that in order to delete a resource, the format of the URI must be */context-root*`/students/`*{id}*. In this example, we are removing the resource (student registration) with identification of 823M934LA (**4**). A transport object is created (**5**) by calling the `axiscTransportCreate()` API and the HTTP method to be used on the HTTP request is set to DELETE (**6**) by calling the `axiscTransportSetProperty()` API. That is it. There is no payload to be sent with the request. The request is send to the server by the call to the subroutine `flushAndReceiveData()` (**7**). More information about this routine is provided later in the article, but basically the subroutine sends the request and handles the response to the request.

The code in Figure 3 shows the logic to create a new student registration record.

Figure 3. Client application code (part 3 of 7)

```
// Now create a new student registration
PRINT ('==Creating student registration record');

uri = 'http://localhost:10035/web/services/students'; 8
axiscTransportReset(tHandle:uri);

propBuf  = 'Content-type' + X'00';
propBuf2 = 'application/json' + X'00'; 9
axiscTransportSetProperty(tHandle: AXISC_PROPERTY_HTTP_HEADER:
                          %addr(propBuf):%addr(propBuf2));

propBuf = 'POST' + X'00';
axiscTransportSetProperty(tHandle: AXISC_PROPERTY_HTTP_METHOD: %addr(propBuf));

request = '{"studentID":"123456789","firstName":'
        + '"New","lastName":"Rec","gender":"Male"}'; 10

rc = axiscTransportSend(tHandle: %ADDR(request): %len(%trim(request)): 0); 11
if (rc = -1);
  checkError ('TransportSend()');
else;
  flushAndReceiveData(); 12
endif;
```

The `axiscTransportReset()` API (**8**) is invoked with the URI that is needed to create a new student registration record. Because JSON data is to be sent, the `axiscTransportSetProperty()` API is invoked to set the content type (**9**) of the HTTP request to application/json, followed by the setting of the HTTP method to POST using the same API. The payload is a JSON formatted request containing the new student registration record (**10**). The data is stored in the transport object by the call to the `axiscTransportSend()` API call (**11**). The request is send to the server by the call to the subroutine, `flushAndReceiveData()` (**12**).

The next step is to retrieve all the student registration records, as shown in Figure 4.

Figure 4. Client application code (part 4 of 7)

```
// Now retrieve all student records
PRINT ('==Retrieving student registration records');
propBuf = 'GET' + X'00'; 13
axiscTransportSetProperty(tHandle: AXISC_PROPERTY_HTTP_METHOD: %addr(propBuf));
flushAndReceiveData(); 14

// Cleanup handle.
axiscTransportDestroy(tHandle); 15

*INLR=*ON;
```

The URI used when creating a new student registration record is used when retrieving the student registration records, and therefore, there is no need to reset the transport object. To retrieve the student registration records, the HTTP method, `GET`, must be used (**13**).

Again, a payload is not required to be sent with the request. The request is send to the server by the call to the subroutine, `flushAndReceiveData()` (**14**). Finally, the transport object is destroyed by the call to the `axiscTransportDestroy()` API (**15**).

Now, let us take a look at the helper subroutines used. Figure 5 shows the `PRINT()` subroutine. The subroutine uses the C runtime `printf()` function that prints to standard output (stdout). So any data passed to the `PRINT()` subroutine is written to standard output.

Figure 5. Client application code (part 5 of 7)

```
// ========================================
// Print to standard out
// ========================================
DCL-PROC PRINT ;
  dcl-pi *n;
    msg varchar(5000) const;
  end-pi;

  dcl-pr printf extproc(*dclcase);
     template pointer value options(*string);
     dummy int(10) value options(*nopass);
  end-pr;

  dcl-c NEWLINE CONST(x'15');

  printf(%TRIM(msg) + NEWLINE);
END-PROC PRINT;
```

Figure 6 shows the `checkError()` subroutine. This subroutine is called if an error occurs when calling a transport API.

Figure 6. Client application code (part 6 of 7)

```
// ========================================
// Handle error
// ========================================
DCL-PROC checkError ;
  dcl-pi *n;
    msg varchar(5000) const;
  end-pi;

  DCL-S axisCode   INT(10);
  DCL-S statusCode POINTER;
  DCL-S rc         INT(10);

  axisCode = axiscTransportGetLastErrorCode(tHandle); 16
  PRINT (msg + ' call failed: ' +
         %CHAR(axisCode) + ':' +
         %STR(axiscTransportGetLastError(tHandle)));

  if (axisCode = EXC_TRANSPORT_HTTP_EXCEPTION); 17
    rc = axiscTransportGetProperty(tHandle:
           AXISC_PROPERTY_HTTP_STATUS_CODE: %ADDR(statusCode));
    PRINT ('HTTP Status code: ' + %STR(statusCode));
  endif;
END-PROC checkError;
```

The subroutine writes the error code and the associated error message (**16**) to the standard output. If the error code indicates that an unexpected HTTP status code (**17**) was returned by the server, the HTTP status code is retrieved and written to the standard output.

Figure 7 shows the `flushAndReceiveData()` subroutine. This subroutine is called to send an HTTP request and receive an HTTP response.

Figure 7. Client application code (part 7 of 7)

```
DCL-PROC flushAndReceiveData;
  dcl-pi *n;
  end-pi;

  DCL-S header     POINTER;
  DCL-S property   CHAR(100);
  DCL-S bytesRead  INT(10) inz(0);

  clear response;
  clear header;

  // Flush data so request is sent
  rc = axiscTransportFlush(tHandle);  18
  if (rc = -1);
    checkError ('TransportFlush()');
    return;
  endif;

  // Receive data and print out data and response to stdout          19
  rc = axiscTransportReceive(tHandle: %ADDR(response): %SIZE(response): 0);
  if (rc = 0);
    PRINT ('No data to read');
  else;
    dow rc > 0 AND bytesRead < %SIZE(response);  20
      bytesRead = bytesRead + rc;
      rc = axiscTransportReceive(tHandle:
                                 %ADDR(response)+bytesRead:
                                 %SIZE(response)-bytesRead:
                                 0);
    enddo;
  endif;
```

The call to the `axiscTransportFlush()` API (**18**) is done to initiate the sending of the HTTP request. The call to the `axiscTransportReceive()` API (**19**) is done to receive the HTTP response to the request. As long as there is data, we loop on the `axiscTransportReceive()` API (**20**) call until there is no data to be consumed.

Seeing the code in action

You can compile the code (assuming that you have loaded and applied the RPG free-form PTFs discussed previously) using the following CL commands (note that *<library>* should be replaced with an existing IBM i library):

```
CRTRPGMOD MODULE(<library>/CLIENTR) SRCSTMF('/clientrest.rpgle')

CRTPGM PGM(<library>/CLIENTR) MODULE(<library>/CLIENTR)
       BNDSRVPGM((QSYSDIR/QAXIS10CC))
```

After the program is created, start a QShell session (using the QSH CL command) and invoke the program by issuing the following command:

```
system 'call <library>/clientr'
```

If you have the web service deployed and running on your system and everything runs successfully, you should see something like what is shown in Figure 8.

Figure 8. Client application code output

```
                                QSH Command Entry

> system 'call amra/clientr'
  ==Deleting student registration record
  No data to read
  HTTP status code: 204
  ==Creating student registration record
  No data to read
  HTTP status code: 201
  ==Retrieving student registration records
  Bytes read: 250
  Data: {"students":[{"studentID":"123456789","firstName":"New","lastName":"Rec","gender":"Male"},{"studentID":"747F023ZX","firstNa
  me":"Jane","lastName":"Amra","gender":"Female"},{"studentID":"826M660CF","firstName":"John","lastName":"Doe","gender":"Male"}]}
  HTTP status code: 200
```

The delete operation was successful and an HTTP status code of 204 indicates success with no content being returned by the server. The creation of a new student registration record was successful and an HTTP status code of 201 indicates the creation of a new resource. Finally, the retrieval of all the registration records was successful indicated by HTTP status code of 200 (OK). You can see the newly created student registration record in the data.

Summary

The new APIs allow you to send user-defined payloads over the HTTP transport. This support enables you to send REST requests or even SOAP requests while controlling exactly what is sent. The APIs will handle the details of the HTTP protocol while allowing you to handle the important details, which is the payload sent and received. We're continually trying to improve the integrated web services experience, and we'd love to hear from you.

Resources

- For everything about the integrated web services support on IBM i see the product web page.

Code Listings

Source code listing for the SRA application

```
**free
Ctl-Opt DFTNAME(RESTCLIENT);
// ************************************************************************
//                                                                        *
//                    IBM Web Services Client for ILE                     *
//                                                                        *
//   FILE NAME:       client.RPGLE                                        *
```

```
//                                                                     *
//   DESCRIPTION:    Source to do REST request using transport APIs    *
//                                                                     *
// ********************************************************************
//  LICENSE AND DISCLAIMER                                             *
//  ----------------------                                             *
//  This material contains IBM copyrighted sample programming source   *
//  code ( Sample Code ).                                              *
//  IBM grants you a nonexclusive license to compile, link, execute,   *
//  display, reproduce, distribute and prepare derivative works of     *
//  this Sample Code.  The Sample Code has not been thoroughly         *
//  tested under all conditions.  IBM, therefore, does not guarantee   *
//  or imply its reliability, serviceability, or function. IBM         *
//  provides no program services for the Sample Code.                  *
//                                                                     *
//  All Sample Code contained herein is provided to you "AS IS"        *
//  without any warranties of any kind. THE IMPLIED WARRANTIES OF      *
//  MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND              *
//  NON-INFRINGMENT ARE EXPRESSLY DISCLAIMED.                          *
//  SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED           *
//  WARRANTIES, SO THE ABOVE EXCLUSIONS MAY NOT APPLY TO YOU.  IN NO   *
//  EVENT WILL IBM BE LIABLE TO ANY PARTY FOR ANY DIRECT, INDIRECT,    *
//  SPECIAL OR OTHER CONSEQUENTIAL DAMAGES FOR ANY USE OF THE SAMPLE   *
//  CODE INCLUDING, WITHOUT LIMITATION, ANY LOST PROFITS, BUSINESS     *
//  INTERRUPTION, LOSS OF PROGRAMS OR OTHER DATA ON YOUR INFORMATION   *
//  HANDLING SYSTEM OR OTHERWISE, EVEN IF WE ARE EXPRESSLY ADVISED OF  *
//  THE POSSIBILITY OF SUCH DAMAGES.                                   *
//                                                                     *
//   <START_COPYRIGHT>                                                 *
//                                                                     *
//   Licensed Materials - Property of IBM                              *
//                                                                     *
//   5770-SS1                                                          *
//                                                                     *
//   (c) Copyright IBM Corp. 2016, 2016                                *
//   All Rights Reserved                                               *
//                                                                     *
//   U.S. Government Users Restricted Rights - use,                    *
//   duplication or disclosure restricted by GSA                       *
//   ADP Schedule Contract with IBM Corp.                              *
//                                                                     *
//   Status: Version 1 Release 0                                       *
//   <END_COPYRIGHT>                                                   *
//                                                                     *
// ********************************************************************

/COPY /QIBM/ProdData/OS/WebServices/V1/client/include/Axis.rpgleinc

DCL-S rc         INT(10);
DCL-S tHandle    POINTER;

DCL-S uri        CHAR(200);
DCL-S response   CHAR(32768);
DCL-S request    CHAR(32768);
DCL-S propBuf    CHAR(100);
DCL-S propBuf2   CHAR(100);
```

```
// -------------------------------------------------------------------
//  Web service logic. The code will attempt to invoke a Web service.
// -------------------------------------------------------------------

// Uncomment to enable trace
// axiscAxisStartTrace('/tmp/axistransport.log': *NULL);

// Set URI in order to delete student
uri = 'http://localhost:10035/web/services/students/823M934LA';

// Create HTTP transport handle.
tHandle = axiscTransportCreate(uri:AXISC_PROTOCOL_HTTP11);
if (tHandle = *NULL);
  PRINT ('TransportCreate() failed');
  return;
endif;

// Delete student registration, ID is part of URI
PRINT ('==Deleting student registration record');

propBuf = 'DELETE' + X'00';
axiscTransportSetProperty(tHandle: AXISC_PROPERTY_HTTP_METHOD:
%addr(propBuf));
flushAndReceiveData();

// Now create a new student registration
PRINT ('==Creating student registration record');

uri = 'http://localhost:10035/web/services/students';
axiscTransportReset(tHandle:uri);

propBuf  = 'Content-type' + X'00';
propBuf2 = 'application/json' + X'00';
axiscTransportSetProperty(tHandle: AXISC_PROPERTY_HTTP_HEADER:
                          %addr(propBuf):%addr(propBuf2));

propBuf = 'POST' + X'00';
axiscTransportSetProperty(tHandle: AXISC_PROPERTY_HTTP_METHOD:
%addr(propBuf));

request = '{"studentID":"123456789","firstName":'
        + '"New","lastName":"Rec","gender":"Male"}';

rc = axiscTransportSend(tHandle: %ADDR(request): %len(%trim(request)):
0);
if (rc = -1);
  checkError ('TransportSend()');
else;
  flushAndReceiveData();
endif;

// Now retrieve all student records
PRINT ('==Retrieving student registration records');
propBuf = 'GET' + X'00';
axiscTransportSetProperty(tHandle: AXISC_PROPERTY_HTTP_METHOD:
%addr(propBuf));
flushAndReceiveData();
```

```
// Cleanup handle.
axiscTransportDestroy(tHandle);

*INLR=*ON;


// ========================================
// Print to standard out
// ========================================
DCL-PROC PRINT ;
  dcl-pi *n;
    msg varchar(5000) const;
  end-pi;

  dcl-pr printf extproc(*dclcase);
     template pointer value options(*string);
     dummy int(10) value options(*nopass);
  end-pr;

  dcl-c NEWLINE CONST(x'15');

  printf(%TRIM(msg) + NEWLINE);
END-PROC PRINT;

// ========================================
// Handle error
// ========================================
DCL-PROC checkError ;
  dcl-pi *n;
    msg varchar(5000) const;
  end-pi;

  DCL-S axisCode   INT(10);
  DCL-S statusCode POINTER;
  DCL-S rc         INT(10);

  axisCode = axiscTransportGetLastErrorCode(tHandle);
  PRINT (msg + ' call failed: ' +
         %CHAR(axisCode) + ':' +
         %STR(axiscTransportGetLastError(tHandle)));

  if (axisCode = EXC_TRANSPORT_HTTP_EXCEPTION);
    rc = axiscTransportGetProperty(tHandle:
            AXISC_PROPERTY_HTTP_STATUS_CODE: %ADDR(statusCode));
    PRINT ('HTTP Status code: ' + %STR(statusCode));
  endif;
END-PROC checkError;

// ========================================
// Flush and Receive data
// ========================================
DCL-PROC flushAndReceiveData;
  dcl-pi *n;
  end-pi;

  DCL-S header     POINTER;
```

```
  DCL-S property   CHAR(100);
  DCL-S bytesRead  INT(10) inz(0);

  clear response;
  clear header;

  // Flush data so request is sent
  rc = axiscTransportFlush(tHandle);
  if (rc = -1);
    checkError ('TransportFlush()');
    return;
  endif;

  // Receive data and print out data and response to stdout
  rc = axiscTransportReceive(tHandle: %ADDR(response): %SIZE(response):
0);
  if (rc = 0);
    PRINT ('No data to read');
  else;
    dow rc > 0 AND bytesRead < %SIZE(response);
      bytesRead = bytesRead + rc;
      rc = axiscTransportReceive(tHandle:
                                 %ADDR(response)+bytesRead:
                                 %SIZE(response)-bytesRead:
                                 0);
    enddo;
  endif;

  if (rc = -1);
    checkError ('TransportReceive()');
  elseif (bytesRead  > 0);
    PRINT ('Bytes read: ' + %CHAR(bytesRead));
    PRINT ('Data: ' + response);
  endif;

  if (rc > -1);
    rc = axiscTransportGetProperty(tHandle:
                                   AXISC_PROPERTY_HTTP_STATUS_CODE:
                                   %addr(header));
    if (rc = -1);
      checkError ('TransportGetProperty()');
    else;
      PRINT ('HTTP status code: ' + %str(header));
    endif;
  endif;

END-PROC flushAndReceiveData;
```