

# Options for Consuming REST APIs



## from RPG

Presented by

Scott Klement

<http://www.scottklement.com>

© 2019-2023, Scott Klement

Helium walks into a bar and orders a beer, the bartender says,  
“Sorry, we don’t serve noble gases here.”  
*(He doesn’t react.)*

## Our Agenda



1. Quick Introduction to REST APIs
  - What are they
  - How they work
  - Example with a testing tool
2. Calling from RPG
  - Free options available
  - Demonstration of the methods
3. Comparing the RPG options
  - Ease of use?
  - Full-featured?
  - Performance?
  - Installation?

## What is a REST API?



An API call using internet-type communications

- "API" refers to a program that has no user interface and is meant to be called by other programs
- Input comes from "parameters"
- Output is returned in "parameters"
- They provide a "service" for their caller
- Can be called on the local machine, LAN, WAN or Internet (at provider's discretion)

3

## Like Calling a Program



If you were to do that using a standard program call, it'd look like this:

```
CALL PGM(MYPROGRAM) PARM(&inputparm &outputparm)
```

A REST API does this using a URL instead of a program name and documents (typically JSON or XML) instead of parameters. It may help to think of it like this:

```
CALL PGM(http://host/myprogram/key) PARM(&InputJSON &OutputJSON)
```

That's just the concept, of course. You can't really use the CALL command to call a URL!

4

# Internet-type Communications



- I really meant "HTTP".
- A REST API is a type of "web service" (or "microservice")
- The the *only* "web" part about "web services" is that they use HTTP.
- Is *not* the same as a web page (does not have a UI)
- *A web browser is not used.*
- Can be consumed by a web page but doesn't have to be!
- Can be a green-screen application, mobile application, Windows application, etc.
- *Always platform/language agnostic.*  
Can be called from anywhere.

5

# Watson Translation Example



IBM Watson provides a simple REST API to translate text from English ("en") to Spanish ("es", short for Español).

URL is: <http://gateway.watsonplatform.net/language-translator/api/v3/translate?version=2018-05-01>

Input parameters are in JSON format (one document, multiple parameters):

```
{
  "source": "en",
  "target": "es",
  "text": [ "Hello" ]
}
```

I wanted to translate the word "Hello"

Output is also JSON:

```
{
  "translations": [{
    "translation": "Hola"
  }],
  "word_count": 1,
  "character_count": 5
}
```

Watson tells me that it's "Hola" in Spanish.

6

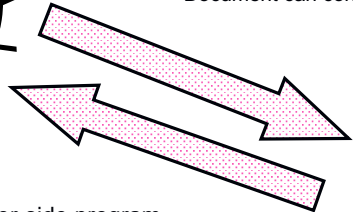
## How Does It Really Work?



HTTP starts with a request for the server

- Can include a document (XML, JSON, etc)
- Document can contain "input parameters"

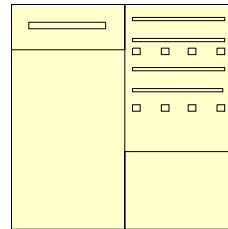
```
{  
  "source": "en",  
  "target": "es",  
  "text": [ "Hello" ]  
}
```



HTTP then runs server-side program

- Sends input doc, waits for program completion
- Returns an output document (XML, JSON, etc)

```
{  
  "translations": [{  
    "translation": "Hola"  
  }],  
  "word_count": 1,  
  "character_count": 5  
}
```



7

## Consuming vs. Providing



In Web Services, these terms are important:

- **Provider** = program that provides a service (the "server" side of communications). This is the API.
- **Consumer** = program that makes the call (the "client" side of communications.). This calls the API.

*This session focuses on consuming (not providing) web services.*

8

## How Can We Try It Out?



- APIs are meant for program-to-program communication
- Normally, to use them, you must write a program!
- A web service testing tool allows testing without writing a program.
  - Postman <http://www.getpostman.com> (REST GUI)
  - SoapUI <http://www.soapui.com> (SOAP/REST GUI)
  - CURL <https://curl.haxx.se/> (command-line driven)

You wouldn't use a testing tool in a production scenario, but they're very useful for making sure the API works

9

## Setting It Up in SoapUI



SoapUI 5.4.0

File Project Suite Case Step Tools Desktop Help

Empty SOAP REST Import Save All Forum Trial Preferences Proxy

New REST Project

New REST Project  
Creates a new REST Project in this workspace

URL:

OK Cancel Import WADL...

- Use a REST web service.
- Provide the URL from IBM Cloud for the Language Translator

Note: This URL is too long to appear on the screen, but the box scrolls left/right to fit it all.

The full URL is

<http://gateway.watsonplatform.net/language-translator/api/v3/translate?version=2018-05-01>

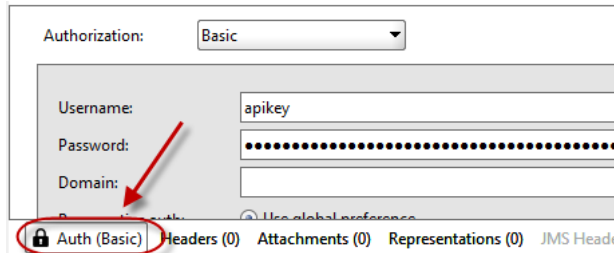
10

# Authorizing SoapUI



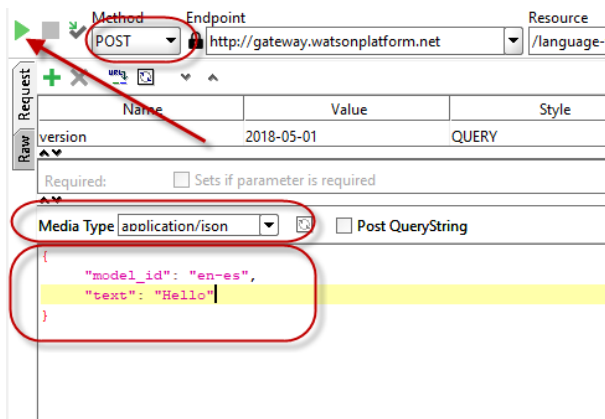
Watson requires you to have an account set up on IBM Cloud that is used to run this service.

In SoapUI you can put your login credentials (usually 'apikey' for the userid plus your password) under 'Auth' at the bottom.



11

# Trying It Out in SoapUI



- Use the "method" dropdown to pick "POST"
- Make sure the media type is "application/json"
- Type the parameters in JSON format into the box
- Click the green "Play" button (upper-left) to run it.

12

# Results



```
Raw | HTML | JSON | XML
1  {
2  "translations": [{"translation": "Hola"}],
3
4  "word_count": 1,
5  "character_count": 5
}
```

- On the right you have tabs to view the result as "Raw", "HTML", "JSON" or "XML"
- Watson services use JSON (as do most newer APIs)
- The result is shown in the box.

# What Might This Look Like from RPG?



For example, the data from this screen can be fed into the code from the last slide.  
The output of the last slide can be placed under "To Text".

```
Translate Text with IBM Watson
Languages: en to es  EN=English ES=Spanish FR=French IT=Italian PT=Portuguese
From Text:
Hello, my name is Scott█
█
█
█
█
█
To Text:
Hola, mi nombre es Scott

HTTP Code:
F3=Exit
5250                                024/006
```

## Challenges To Overcome



### What problems would we need to solve to do it from RPG?

- Tool to create a JSON (or XML) document
- Some way to do http:// URL from RPG
- Tool to read the returned JSON (or XML)

### Other things we might need?

- Ability to specify different methods (GET, POST, PUT, DELETE, etc)
- Somehow specify login credentials
- Transport Layer Security (TLS, often called by the older name “SSL”)
- Sometimes custom HTTP headers are needed
- Encodings sometimes needed (depending on the API)
  - XML and JSON, of course
  - URL-encoding (aka “web page forms”)
  - Multipart documents (aka “attachments”)
  - Base64 encoding

15

## Free Options Available



### Free Options Available for RPG

- Open Source **HTTPAPI**
- Two sets of IBM-supplied **SQL** routines (SYSTOOLS & QSYS2)
- IBM-supplied **AXIS** routines

### Other Languages

- Java, PHP, Ruby, Python, Node.js all provide options, here.

### Commercial Options

- Various vendors provide tools. (example: Bradley Stone’s GETURI)
- I’m not familiar with all of the options available

16



# HTTAPI



## Open Source (completely free tool)

- Created by Scott Klement, originally in 2001
- Written in native RPG code
- <http://www.scottklement.com/httapi>

## Provides Routines For

- HTTP and HTTPS (TLS/SSL) communications
- URL (web form) encoding
- Multipart (attachment) encoding
- Basic, Digest and NTLM2 authentication

## Usually Used With Other Open Source Tools

- **Expat** for reading XML (or use XML-INTO) <http://scottklement.com/expat>
- **YAJL** for reading/writing JSON (works with DATA-INTO) <http://scottklement.com/yajl>
- **BASE64** tool <http://scottklement.com/base64>

17

# http\_string syntax



## Making HTTP Requests

- **http\_req** = general-purpose HTTP request, lots of options
- **http\_stmf** = simplified HTTP request, where data is read from/written to IFS files
- **http\_string** = simplified HTTP request where data is read/written from/to RPG strings

*data-received* = **http\_string**( *method* : *url* : *string-to-send* : *content-type* )

- **method** = HTTP method (GET, POST, PUT, DELETE, etc)
- **url** = The URL to communicate with
- **string-to-send** = RPG char/varchar string to send to URL
- **content-type** = Internet media type (MIME type) of data you're sending
- **data-received** = RPG char/varchar string to contain data returned from server

## Other Routines

- **http\_setAuth** = set authentication (user/password)
- **http\_setOption** = set various options
- **http\_error** = retrieve error code, message, and http status code

18

# Language Translation in RPG



```
http_setAuth( HTTP_AUTH_BASIC: 'apikey': '{your-api-key}');  
request = '{"source":"en","target":"es","text":["Hello"]}';  
url = 'https://gateway.watsonplatform.net/language-translator/api'  
      + '/v3/translate?version=2018-05-01'  
response = http_string('POST': url: request: 'application/json');  
DATA-INTO result %DATA(response) %PARSER('YAJLINTO');
```

http\_setAuth() – sets the userid/password used.

http\_string() – sends an HTTP request, getting the input/output from strings

DATA-INTO – RPG opcode for parsing documents such as JSON

**request**, **url** and **response** are standard RPG VARCHAR fields. (CHAR would also work)

19

# Quick JSON/XML Primer



```
D list          ds          qualified  
D              ds          dim(2)  
D custno       4p 0  
D name         25a
```

Array of data structures  
in RPG...

```
[  
  {  
    "custno": 1000,  
    "name": "ACME, Inc"  
  },  
  {  
    "custno": 2000,  
    "name": "Industrial Supply Limited"  
  }  
]
```

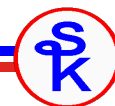
Array of data structures  
in JSON

```
<list>  
  <cust>  
    <custno>1000</custno>  
    <name>Acme, Inc</name>  
  </cust>  
  <cust>  
    <custno>2000</custno>  
    <name>Industrial Supply Limited</name>  
  </cust>  
</list>
```

Array of data structures  
in XML

20

## Reading JSON with YAJL (DATA-INTO)



```
dcl-ds result qualified;           // {
dcl-ds translations dim(1);        //  "translations": [{
  translation varchar(1000);      //    "translation": "Hola"
end-ds;                            //  }],
word_count int(10);               //  "word_count": 1,
character_count int(10);          //  "character_count": 5
end-ds;                            // }

data-into result %DATA(response) %PARSER('YAJLINTO');
```

- Based on XML-INTO
- Copy data from JSON into a matching RPG data structure
- JSON {} symbols start/end an object ("data structure")
- JSON [] symbols start/end an array
- RPG definition should match exactly (including field names, nesting, etc)
- Now data is in an RPG DS that's easy for an RPGer to work with

21

## Creating JSON with YAJL (DATA-GEN)



```
dcl-s request varchar(2000);

dcl-ds reqds qualified;           // {
  source varchar(2);              //  "source": "{string}",
  target varchar(2);              //  "target": "{string}",
  text varchar(1000) dim(1);      //  "text": [ "{string}" ]
end-ds;                            // }

reqds.source = 'en';
reqds.target = 'es';
reqds.text(1) = 'Hello';

data-gen reqds %data(request) %gen('YAJLDTAGEN');
```

```
{
  "source": "en",
  "target": "es",
  "text": [ "Hello" ]
}
```

- Like DATA-INTO, but in reverse... DATA-GEN creates the JSON
- Format of JSON comes from the RPG data structure
- Resulting JSON string is placed in 'request' variable.

22

## HTTPAPI Example (1 of 5)



To put all of these concepts together, here's the full RPG code for the translate example, using HTTPAPI and YAJS.

```
**free
ctl-opt option(*srcstmt) dftactGrp(*no)
      bnddir('HTTPAPI':'YAJS');

/copy httpapi_h
/copy yaaj_h

dcl-f WATSONR3D workstn indds(dspf);

dcl-Ds dspf qualified;
      F3Exit ind pos(3);
end-Ds;

dcl-c UPPER 'ENESFRITPT';
dcl-c lower 'enesfritpt';

fromLang = 'en';
toLang   = 'es';
```

BNDDIR is used to bind your program to the tools

Copybooks contain the definitions we'll need to call the HTTPAPI and YAJS routines.

23

## HTTPAPI Example (2 of 5)



Main loop controls the flow of the program, repeating the screen until F3 key is pressed.

```
dou dspf.F3Exit = *on;

  exfmt screen1;
  if dspf.F3exit = *on;
    leave;
  endif;

  fromLang = %xlate(UPPER:lower:fromLang);
  toLang   = %xlate(UPPER:lower:toLang);
  toText = translate( fromLang: toLang: %trim(fromText) );

enddo;

*inlr = *on;
return;
```

the translate procedure is what actually calls Watson

24

## HTTPAPI Example (3 of 5)



```
dcl-proc translate;

dcl-pi *n varchar(1000);
  fromLang char(2)      const;
  toLang   char(2)      const;
  fromText varchar(1000) const;
end-pi;

dcl-s url      varchar(2000);
dcl-s request  varchar(2000);
dcl-s response varchar(5000);

dcl-ds result qualified;
  dcl-ds translations dim(1);
    translation varchar(1000);
  end-ds;
  word_count int(10);
  character_count int(10);
end-ds;
```

Most of this slide is just ordinary RPG definitions

← Data structure must match the JSON format for the output parameters.

25

## HTTPAPI Example (4 of 5)



```
dcl-ds reqds qualified;           // {
  source varchar(2);             //   "source": "{string}",
  target varchar(2);             //   "target": "{string}",
  text  varchar(1000) dim(1);     //   "text": [ "{string}" ]
end-ds;                           // }

// Generate the JSON document to send

reqds.source = fromLang;
reqds.target = toLang;
reqds.text(1) = fromText;

data-gen reqds %data(request) %gen('YAJLDAGEN');
```

← Data structure must match the JSON format for the input parameters.

↑ Result placed in request variable

26

## HTTPAPI Example (5 of 5)



27

## Error Handling with HTTPAPI



http\_string throws an exception if there's an error. If you don't mind the user receiving an exception when something goes wrong, you can code as follows (and let the OS handle it.)

```
response = http_string('POST': url: request: 'application/json');
```

To handle it yourself, use RPG's monitor/on-error opcodes.

```
monitor;
  response = http_string('POST': url: request: 'application/json');
on-error;
  errorMsg = http_error();
endmon;
```

http\_error() returns the last error message. You can also use it to get the last error number and HTTP status code by passing optional parameters.

```
dcl-s msg varchar(100);
dcl-s errnum int(10);
dcl-s status int(10);

msg = http_error( errnum : status );
```

28

# SQL QSYS2 HTTP Functions



## Included in IBM's QSYS2 schema (library)

- Added in September 2021 (7.3 TR11, 7.4 TR5, 7.5 at GA)
- Updated in subsequent TRs and group PTFs
- The best part? Nothing to install!
- The next best? Easy to use!

## Unlike SYSTOOLS, Doesn't Use Java!!

- Therefore HTTP\_POST is much faster than HTTPPOSTCLOB (same for other similar operations, HTTP\_GET, HTTP\_PUT, HTTP\_DELETE run better than HTTPxxxCLOB versions.)
- Need a "real" CCSID. Your job should not be 65535. This is because data is sent/received in Unicode

## Provides:

- HTTP routines
- Routines for reading/writing XML/JSON
- URLENCODE and BASE64 routines

29

# SQL Functions Available



## HTTP Routines

- HTTP\_GET(), HTTP\_POST, HTTP\_PUT(), HTTP\_DELETE(), HTTP\_PATCH()
- HTTP\_GET\_VERBOSE(), HTTP\_POST\_VERBOSE(), HTTP\_PUT\_VERBOSE(), HTTP\_DELETE\_VERBOSE(), HTTP\_PATCH\_VERBOSE()

## JSON/XML Routines

- JSON\_TABLE
- JSON\_OBJECT, JSON\_ARRAY, et al
- XMLTABLE
- BASE64ENCODE or BASE64DECODE
- URLENCODE or URLDECODE

<https://www.ibm.com/docs/en/i/7.5?topic=programming-http-functions-overview>

30

## Same Example with SQL



Included in IBM's QSYS2 schema (library)

- No need to rewrite whole program
- Just re-write the `translate()` subprocedure.

We need to

- Create a JSON object (`JSON_OBJECT` function) as a character string
- Send the character string via HTTP POST method (`HTTP_POST`)
- Receive the response as a character string
- Interpret the received JSON string (`JSON_TABLE`)

NOTE:

- Its not required that we use the SQL JSON together with the SQL HTTP routines
- We could use YAJL for JSON and SQL for HTTP
- Or SQL for JSON and HTTPAPI for HTTP
- etc.

31

## HTTP\_POST Syntax



`HTTP_POST` is an SQL function (UDF) you can call from within another SQL statement. (Typically a `VALUES` or `SELECT` statement.)

`HTTP_POST` ( *url* , *requestMessage* , *options* )

- *url* = an expression containing the URL to connect to
- *requestMessage* = an expression containing the message to send
- *options* = a string expression (formatted as JSON) containing options that control the request.

**Returns:** A CLOB(2g) CCSID 1208 containing the response from the server

**Note:** All of the above are UTF-8 (CCSID 1208). SQL will automatically perform conversions, so be sure your job CCSID is set properly.

For example, the EBCDIC typically used in the USA is CCSID 37. If your QCCSID system value isn't set properly, you can override it temporarily in the job like this:

`CHGJOB CCSID(37)`

32



# SQL HTTP Options



## Options are

- Formatted as JSON
- If an option has multiple parameters, they are separated with commas

"option-name": "option parameter 1,option parameter 2"

```
{
  "basicAuth": "MyUserId,MyPassword",
  "connectTimeout": 180,
  "header": "Content-type,application/json; charset=UTF-8",
  "header": "Accept,application/json,*",
  "redirect": 5
}
```

## Some options are:

- basicAuth = userid/password needed to log in with basic authentication
- connectTimeout = seconds to wait for connection before timing out
- redirect = number of times to follow a redirect before failing
- header = HTTP header to include (may be specified multiple times)

All options are documented here:

[https://www.ibm.com/docs/en/i/7.4?topic=functions-http-get#rbafzsahttpget\\_HTTP\\_options](https://www.ibm.com/docs/en/i/7.4?topic=functions-http-get#rbafzsahttpget_HTTP_options)

33

# Simple HTTP\_POST Example:



```
request = '{ "test": "json" }';

url = 'https://gateway.watsonplatform.net/language-translator/api'
      + '/v3/translate?version=2018-05-01';

options = '{ "basicAuth": "apikey,my-password-here", +
            "header": "content-type,application/json" }';

exec SQL
  values QSYS2.HTTP_POST(:url, :request, :options)
  into :response;
```

## This will

- Connect to the given URL
- Log in as userid=apikey, password=my-password-here
- Tell the server at the URL to expect data in application/json format
- Send the (mocked up example) JSON
- Receive the response into the "response" variable

34

## SQL JSON Publishing (1 of 2)



Create a JSON object:

```
JSON_OBJECT( KEY 'name' VALUE 'val', KEY 'name2' VALUE 'val2')
```

```
JSON_OBJECT( 'name' VALUE 'val', 'name2' VALUE 'val2' )
```

```
JSON_OBJECT( 'name': 'val', 'name2': 'val2' )
```

Result:

```
{ "name": "val", "name2": "val2" }
```

- The three syntaxes all do the same thing. (The word KEY is optional, and the word VALUE can be replaced with a colon.)
- Instead of a character string, the value can be a number, another json object, or a json array.
- Remember: These are SQL functions, used within an SQL statement.

35

## SQL JSON Publishing (2 of 2)



Create a JSON array:

```
JSON_ARRAY( 'val1', 'val2', etc )
```

```
JSON_ARRAY( full-select )
```

Result:

```
[ "val1", "val2", "val3" ]
```

- Instead of a character string, the values can be numbers or other json object/arrays
- The full-select is an SQL select statement. It must return only a single column.
- If one full-select is given, it may return multiple rows. Each row becomes its own array entry.
- It's possible to list multiple select statements or combine them with values. In that case, the select statement must return only one row.

36

# SQL Reading JSON



JSON\_TABLE is an SQL table function (UDTF)

This is meant to read a JSON document and treat the output as an SQL table, allowing you to query it, use it in a program, etc.

**JSON\_TABLE**( *json-document*, *path* COLUMNS( *column-definitions* ) )

- *json-document* = the json document as a char, varchar, clob, etc
- *path* = path within the JSON document to be read
- *column-definitions* = defines each column and how to retrieve it

```
1 SELECT J."id", J."name", J."postal"
2   FROM JSON_TABLE( '{ "id": 501, "name": "Test Customer", "address": { "postal": "98765" } }',
3                    'lax 0'
4                    COLUMNS(
5                      "id" DECIMAL(4, 0),
6                      "name" VARCHAR(25),
7                      "postal" VARCHAR(10) PATH 'lax 0.address.postal'
8                    )
9                ) AS J;
```

id	name	postal
501	Test Customer	98765

37

# SQL HTTP Example (1 of 4)



```
dcl-proc translate;
```

```
dcl-pi *n varchar(1000);
      fromLang char(2)      const;
      toLang char(2)        const;
      fromText varchar(1000) const;
end-pi;
```

Just some definitions here...

```
dcl-s userid  varchar(10);
dcl-s password varchar(200);
dcl-s url     varchar(2000);
dcl-s request varchar(2000);
dcl-s response varchar(5000);
dcl-s retval  varchar(1000);
dcl-s options varchar(1000);
```

38

## SQL HTTP Example (2 of 4)



```
exec sql values json_object(
    'source' value lower(:fromLang),
    'target' value lower(:toLang),
    'text' value json_array(:fromText)
)
into :request;
if %subst(sqlstt:1:2) <> '00' and %subst(sqlstt:1:2) <> '01';
// Handle error
endif;

userid = 'apikey';
password = 'password';

exec sql values json_object(
    'basicAuth' value :userid || ',' || :password,
    'header' value 'Content-Type,application/json'
)
into :options;
if %subst(sqlstt:1:2) <> '00' and %subst(sqlstt:1:2) <> '01';
// Handle error
endif;
```

```
{
  "source": "en",
  "target": "es",
  "text": [ "Hello" ]
}
```

```
{
  "basicAuth": "apikey,password",
  "header": "Content-type,application/json"
}
```

39

## SQL HTTP Example (3 of 4)



```
url = 'https://+
      api.us-south.language-translator.watson.cloud.ibm.com+
      /instances/66f38a33-6f74-492a-8025-8a2e1759a228+
      /v3/translate?version=2018-05-01';

exec SQL
  values QSYS2.HTTP_POST(:url, :request, :options)
  into :response;

if %subst(sqlstt:1:2) <> '00' and %subst(sqlstt:1:2) <> '01';
  retval = '**ERROR IN HTTP_POST: SQLSTT=' + sqlstt;
  return retval;
endif;
```

This will

- Connect/Login with the options from the previous slide
- Send the JSON document created on the previous slide
- Return the output from the server into "response"

40

## SQL HTTP Example (4 of 4)



```
exec SQL SELECT J."translation"
into :retval
from JSON_TABLE(:response, 'lax $'
  COLUMNS(
    "translation" VARCHAR(1000)
    PATH 'lax $.translations[0].translation'
  )
) as J;

if %subst(sqlstt:1:2) <> '00' and %subst(sqlstt:1:2) <> '01';
  retval = '** ERROR READING JSON: SQLSTT=' + sqlstt;
  return retval;
endif;

return retval; // Will contain: Hola

end-proc;
```

```
{
  "translations": [{
    "translation": "Hola"
  }],
  "word_count": 1,
  "character_count": 5
}
```

41

## Alternately, Combine All SQL Into One Statement



```
exec SQL SELECT J."translation"
into :retval
from JSON_TABLE(
  HTTP_POST(
    :url,
    json_object(
      'source' value lower(:fromLang),
      'target' value lower(:toLang),
      'text' value json_array(:fromText)
    ),
    json_object(
      'basicAuth' value :userid || ',' || :password,
      'header' value 'Content-Type,application/json'
    )
  ),
  'lax $' COLUMNS(
    "translation" VARCHAR(1000)
    PATH 'lax $.translations[0].translation'
  )
) as J;
if %subst(sqlstt:1:2) <> '00' and %subst(sqlstt:1:2) <> '01';
  retval = '** ERROR CALLING API: SQLSTT=' + sqlstt;
  return retval;
endif;
```

42

# Db2 SYSTOOLS (Older SQL)



## Included in IBM's SYSTOOLS schema (library)

- First added in 2014, just after IBM i 7.2 release.
- Updated several times in Technology Refreshes for 7.1+
- The best part? Nothing to install!
- The next best? Easy to use!

## Uses Java Under the Covers

- You must have a JVM (1.6 or newer) installed
- Starts the JVM in each job (performance considerations)
- Need a "real" CCSID. Your job should not be 65535.

## Provides:

- HTTP routines
- Routines for reading/writing XML/JSON
- URLENCODE and BASE64 routines

43

# SQL Functions in SYSTOOLS



## HTTP Routines

- HTTPxxxBLOB or HTTPxxxCLOB functions (*xxx can be GET, POST, PUT or DELETE*)
- HTTPBLOB or HTTPCLOB functions
- HTTPxxxBLOBVERBOSE or HTTPxxxCLOBVERBOSE table functions
- HTTPHEAD

## JSON/XML Routines

- JSON\_TABLE
- JSON\_OBJECT, JSON\_ARRAY, et al
- XMLTABLE
- BASE64ENCODE or BASE64DECODE
- URLENCODE or URLDECODE

[https://www.ibm.com/support/knowledgecenter/ssw\\_ibm\\_i\\_74/rzajq/rzajqudfhttpclob.htm](https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/rzajq/rzajqudfhttpclob.htm)

44

## Same Example with SYSTOOLS



Included in IBM's SYSTOOLS schema (library)

- No need to rewrite whole program
- Just re-write the `translate()` subprocedure.

We need to

- Create a JSON object (`JSON_OBJECT` function) as a character string
- Send the character string via HTTP POST method (`HTTPPOSTCLOB`)
- Receive the response as a character string
- Interpret the received JSON string (`JSON_TABLE`)

NOTE:

- Its not required that we use the SQL JSON together with the SQL HTTP routines
- We could use YAJL for JSON and SQL for HTTP
- Or SQL for JSON and HTTPAPI for HTTP
- etc.

45

## HTTPPOSTCLOB Syntax



HTTPPOSTCLOB is an SQL function (UDF) you can call from within another SQL statement. (Typically a SELECT statement.)

`HTTPPOSTCLOB( url, headersXML, requestMessage )`

- `url` = a varchar(2048) containing the URL to connect to
- `headersXML` = a CLOB(10k) containing an XML document that specifies any custom HTTP headers. (Can be null if you don't wish to customize the headers)
- `requestMessage` = a CLOB(2G) containing the message to send

Returns: A CLOB(2g) containing the response from the server

**Note:** All of the above are UTF-8 (CCSID 1208). SQL will automatically perform conversions, so be sure your job CCSID is set properly.

For example, the EBCDIC typically used in the USA is CCSID 37. If your QCCSID system value isn't set properly, you can override it temporarily in the job like this:

`CHGJOB CCSID(37)`

46

## SQL JSON Publishing (1 of 2)



Create a JSON object:

```
JSON_OBJECT( KEY 'name' VALUE 'val', KEY 'name2' VALUE 'val2')
```

```
JSON_OBJECT( 'name' VALUE 'val', 'name2' VALUE 'val2' )
```

```
JSON_OBJECT( 'name': 'val', 'name2': 'val2' )
```

Result:

```
{ "name": "val", "name2": "val2" }
```

- The three syntaxes all do the same thing. (The word KEY is optional, and the word VALUE can be replaced with a colon.)
- Instead of a character string, the value can be a number, another json object, or a json array.
- Remember: These are SQL functions, used within an SQL statement.

47

## SQL JSON Publishing (2 of 2)



Create a JSON array:

```
JSON_ARRAY( 'val1', 'val2', etc )
```

```
JSON_ARRAY( full-select )
```

Result:

```
[ "val1", "val2", "val3" ]
```

- Instead of a character string, the values can be numbers or other json object/arrays
- The full-select is an SQL select statement. It must return only a single column.
- If one full-select is given, it may return multiple rows. Each row becomes its own array entry.
- It's possible to list multiple select statements or combine them with values. In that case, the select statement must return only one row.

48



# SQL Reading JSON



JSON\_TABLE is an SQL table function (UDTF)

This is meant to read a JSON document and treat the output as an SQL table, allowing you to query it, use it in a program, etc.

**JSON\_TABLE**( *json-document*, *path* COLUMNS( *column-definitions* ) )

- *json-document* = the json document as a char, varchar, clob, etc
- *path* = path within the JSON document to be read
- *column-definitions* = defines each column and how to retrieve it

```
1 SELECT J."id", J."name", J."postal"
2   FROM JSON_TABLE( '{ "id": 501, "name": "Test Customer", "address": { "postal": "98765" } }',
3                    'lax 0'
4                   COLUMNS(
5                       "id" DECIMAL(4, 0),
6                       "name" VARCHAR(25),
7                       "postal" VARCHAR(10) PATH 'lax 0.address.postal'
8                   )
9                   ) AS J;
```

id	name	postal
501	Test Customer	98765

49

# Db2 SQL Example (1 of 4)



```
dcl-proc translate;
```

```
dcl-pi *n varchar(1000);
  fromLang char(2)      const;
  toLang char(2)        const;
  fromText varchar(1000) const;
end-pi;
```

```
dcl-s userid  varchar(10);
dcl-s password varchar(200);
dcl-s hdr     varchar(200);
dcl-s url     varchar(2000);
dcl-s request varchar(2000);
dcl-s response varchar(5000);
dcl-s retval  varchar(1000);
```

Most of this slide is  
just ordinary RPG  
definitions

50

## Db2 SQL Example (2 of 4)



```
exec sql select json_object(
    'source' value :fromLang,
    'target' value :toLang,
    'text' value json_array(:fromText)
)
into :request
from SYSIBM.SYSDUMMY1;
```

```
if %subst(sqlstt:1:2) <> '00' and %subst(sqlstt:1:2) <> '01';
retval = '**ERROR CREATING: SQLSTT=' + sqlstt;
return retval;
endif;
```

Error checking is done the same as any other SQL statement.

```
json_object(
'source' value 'en',
'target' value 'es',
'text' value json_array('Hello')
)
```



```
{
  "source": "en",
  "target": "es",
  "text": [ "Hello" ]
}
```

51

## Db2 SQL Example (3 of 4)



```
userid = 'apikey';
password = 'your-Watson-api-key-goes-here';

url = 'https://' + userid + ':' + password + '@'
+ 'gateway.watsonplatform.net/language-translator/api'
+ '/v3/translate?version=2018-05-01';
```

The easiest way to do user/password is add them to the URL.

```
hdr = '<httpHeader>+
<header name="Content-Type" value="application/json" />+
</httpHeader>';
```

```
exec SQL
select SYSTOOLS.HTTPPOSTCLOB(:url, :hdr, :request)
into :response
from SYSIBM.SYSDUMMY1;
```

```
if %subst(sqlstt:1:2) <> '00' and %subst(sqlstt:1:2) <> '01';
retval = '**ERROR IN HTTP: SQLSTT=' + sqlstt;
return retval;
endif;
```

Error checking is done the same as any other SQL statement.

52

## Db2 SQL Example (4 of 4)



```
exec SQL SELECT J."translation"
into :retval
from JSON_TABLE(:response, 'lax $'
COLUMNS(
"translation" VARCHAR(1000)
PATH 'lax $.translations[0].translation'
)
) as J;

if %subst(sqlstt:1:2) <> '00' and %subst(sqlstt:1:2) <> '01';
retval = '** ERROR READING: SQLSTT=' + sqlstt;
return retval;
endif;

return retval;

end-proc;
```

JSON\_TABLE is a syntax for mapping JSON into a virtual table. Once it is viewed as a table, you can SELECT INTO to get it into an RPG variable.

53

## Error Handling with Db2 SQL



Since the HTTP, JSON, XML, etc functions in Db2 are simply SQL statements, you can tell if something failed by checking SQLSTATE (SQLSTT) or SQLCODE (SQLCOD) the same as you would a regular SQL statement.

```
exec SQL (any SQL statement here);

if %subst(sqlstt:1:2) <> '00' and %subst(sqlstt:1:2) <> '01';
retval = '** SQL ERROR: SQLSTT=' + sqlstt;
return retval;
endif;
```

However, this does not provide a lot of detail about the problem.

Calling the VERBOSE table functions (example: HTTPPOSTCLOBVERBOSE) does provide a little more information but does not provide in-depth diagnostics.

For example, if you provide an invalid URL, you simply get back a null.

But, if you connect to a valid host and it returns "404 Not Found" you can get that message from the VERBOSE function.

54

# Db2 SQL HTTP Functions



Links to details for the various SQL functions in the IBM Knowledge Center

SQL HTTP routines:

[https://www.ibm.com/support/knowledgecenter/ssw\\_ibm\\_i\\_74/rzaiq/rzaiqhttpoverview.htm](https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/rzaiq/rzaiqhttpoverview.htm)

JSON\_OBJECT

[https://www.ibm.com/support/knowledgecenter/ssw\\_ibm\\_i\\_74/db2/rbafzscajsonobject.htm](https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/db2/rbafzscajsonobject.htm)

JSON\_ARRAY

[https://www.ibm.com/support/knowledgecenter/ssw\\_ibm\\_i\\_74/db2/rbafzscajsonarray.htm](https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/db2/rbafzscajsonarray.htm)

JSON\_TABLE

[https://www.ibm.com/support/knowledgecenter/ssw\\_ibm\\_i\\_74/db2/rbafzscajsonatable.htm](https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/db2/rbafzscajsonatable.htm)

Don't forget, these won't work if you have sysval QCCSID = 65535 unless you set the CCSID in your job!

```
chgjob ccsid(37)
```

55

# AXIS Transport API



## IBM-supplied

- Comes with the IBM HTTP server, so no need for third-party software
- Runs behind the old wsdl2ws.sh/wsdl2rpg.sh SOAP code
- Designed for C, but IBM provides RPG prototypes
- Shipped with the IWS client code starting in 2008

## Documentation

- <https://www.ibm.com/systems/power/software/i/iws/>
- Under "Documentation", click "Web Services Client for ILE Programming Guide"
- Most of this PDF is aimed at SOAP with IBM's generator.
- Needed Transport APIs are in Chapter 17, under "Transport C APIs"

## IBM-supplied Examples With RPG

- <https://developer.ibm.com/articles/i-send-receive-user-defined-soap-rest-messages-trs/>
- <https://www-01.ibm.com/support/docview.wss?uid=nas8N1022250>

56

# AXIS Routines We Can Call



## AXIS Routines

- [axiscTransportCreate](#) = Create a handle for an HTTP connection
- [axiscTransportDestroy](#) = Destroy connection handle
- [axiscSetProperty](#) = Set properties for use in HTTP handle
- [axiscGetProperty](#) = Get properties from an HTTP handle
- [axiscTransportSend](#) = Connect with HTTP and send data.
- [axiscTransportFlush](#) = Data sent is buffered and may not be completely sent until the buffer is flushed (by calling this API)
- [axiscTransportReceive](#) = Receive results from HTTP. This may return only part of the data; call it repeatedly to get everything.
- [axiscGetLastErrorCode](#) = Retrieve the last error number that occurred
- [axiscGetLastError](#) = Retrieve the last error message that occurred
- [axiscAxisStartTrace](#) = Create detailed trace of HTTP connection to IFS file

*NOTE: The AXIS Transport API does not provide any routines for handling XML, JSON, URL-encoding, Base64 encoding, etc. You would need to use routines from elsewhere.*

57

# AXIS Procedure



To use the AXIS routines, the following is needed:

1. Create a handle.
2. Set properties for:
  - HTTP method (GET, POST, PUT, DELETE)
  - Login credentials (Basic Authentication)
  - Content-Type HTTP Header
  - TLS/SSL options
3. Send data, then flush send buffer
4. Receive data in a loop until there's no more to receive
5. Get the property for the HTTP status code
6. Destroy handle
7. If any of the above returns an error, call the routines to get error number/message.

58

# Same Example with AXIS



Included in IBM's SYSTOOLS schema (library)

- No need to rewrite whole program
- Just re-write the `translate()` subprocedure.
- Except: We need to include the AXIS copybook and bind to the QAXIS10CC service program.

```
CRTBNDDIR BNDDIR(your-Lib/AXIS)
ADDBNDDIRE BNDDIR(your-Lib/AXIS) OBJ((QSYSDIR/QAXIS10CC *SRVPGM))
```

```
ctl-opt option(*srcstmt) dftactGrp(*no)
      bnmdir('AXIS': 'YAJL');

/copy yajl_h
/copy /QIBM/ProdData/OS/WebServices/V1/client/include/Axis.rpgleinc
```

Since AXIS doesn't provide routines to work with JSON documents, we will:

- Use SQL to create the JSON
- Use YAJL with DATA-INTO to read the JSON

59

# AXIS Example (1 of 9)



```
dcl-proc translate;

  dcl-pi *n varchar(1000);
    fromLang char(2) const;
    toLang char(2) const;
    fromText varchar(1000) const;
  end-pi;

  dcl-s userid varchar(10);
  dcl-s password varchar(200);
  dcl-s hdr varchar(200);
  dcl-s url varchar(2000);
  dcl-s request varchar(2000);
  dcl-s response varchar(5000);
  dcl-s rcvBuf char(5000);
  dcl-s rc int(10);
  dcl-s propName char(200);
  dcl-s propVal char(200);
  dcl-s transportHandle pointer;

  dcl-ds result qualified;
    dcl-ds translations dim(1);
      translation varchar(1000) inz('');
    end-ds;
  word_count int(10) inz(0);
  character_count int(10) inz(0);
end-ds;
```

Most of this slide is just ordinary RPG definitions

Data structure must match the JSON format for the output parameters. (Same as earlier examples.)

60

## AXIS Example (2 of 9)



```
exec sql select json_object(
    'source' value :fromLang,
    'target' value :toLang,
    'text' value json_array(:fromText)
)
into :request
from SYSIBM.SYSDUMMY1;

if %subst(sqlstt:1:2) <> '00' and %subst(sqlstt:1:2) <> '01';
return '**ERROR CREATING: SQLSTT=' + sqlstt;
endif;
```

Using SQL To Create  
JSON Document  
(same as previous  
example)

61

## AXIS Example (3 of 9)



```
axiscAxisStartTrace('/tmp/axistransport.log': *NULL);

userid = 'apikey';
password = 'your-watson-api-key-here';

url = 'https://gateway.watsonplatform.net/language-translator/api'
+ '/v3/translate?version=2018-05-01';

transportHandle = axiscTransportCreate(url: AXISC_PROTOCOL_HTTP11);
if (transportHandle = *null);
failWithError(transportHandle: 'axiscTransportCreate');
endif;
```

Create detailed diagnostic  
("trace") log of HTTP  
session

Set up transport to use the  
Watson URL and the HTTP  
1.1 protocol. (This is the  
only supported protocol.)

62

## AXIS Example (4 of 9)



```
propName = 'POST' + x'00';
axiscTransportSetProperty( transportHandle
                          : AXISC_PROPERTY_HTTP_METHOD
                          : %addr(propName));

propName = userid + x'00';
propVal  = password + x'00';
axiscTransportSetProperty( transportHandle
                          : AXISC_PROPERTY_HTTP_BASICAUTH
                          : %addr(propName)
                          : %addr(propVal) );

propName = 'Content-Type' + x'00';
propVal  = 'application/json' + x'00';
axiscTransportSetProperty( transportHandle
                          : AXISC_PROPERTY_HTTP_HEADER
                          : %addr(propName)
                          : %addr(propVal) );

propName = '*SYSTEM' + x'00';
propVal  = x'00';
axiscTransportSetProperty( transportHandle
                          : AXISC_PROPERTY_HTTP_SSL
                          : %addr(propName)
                          : %addr(propVal) );
```

Use the POST method

Set user/password  
using basic auth

Set the content-type  
HTTP header

Tell AXIS to use default  
TLS/SSL settings from  
the \*SYSTEM certificate  
store

63

## AXIS Example (5 of 9)



```
rc = axiscTransportSend( transportHandle
                        : %addr(request: *data)
                        : %len(request)
                        : 0 );

if rc = -1;
    failWithError(transportHandle: 'axiscTransportSend');
endif;

rc = axiscTransportFlush(transportHandle);
if rc = -1;
    failWithError(transportHandle: 'axiscTransportFlush');
endif;
```

The network connection  
begins running here  
The %ADDR and %LEN logic  
converts the 'request' variable  
into pointers for AXIS

Since data is buffered, it  
isn't fully sent until the  
buffer is flushed.

64



## AXIS Example (6 of 9)



```
response = '';  
do while rc < 1;  
  rc = axiscTransportReceive( transportHandle  
                             : %addr(rcvBuf)  
                             : %size(rcvBuf)  
                             : 0 );  
  if rc >= 1;  
    response += %subst(rcvBuf:1:rc);  
  endif;  
enddo;  
if rc = -1;  
  failWithError(transportHandle: 'axiscTransportReceive');  
else;  
  httpCode = getHttpStatus(transportHandle);  
endif;  
axiscTransportDestroy(transportHandle);
```

Data will not be received all at once. Keep calling the receive routine until there's no more data.

After each call, add any new data to the end of the response string.

axiscTransportDestroy cleans up the transport when you're done.

65

## AXIS Example (7 of 9)



```
if %len(response) > 0;  
  data-into result %DATA(response) %PARSER('YAJLINTO');  
endif;  
return result.translations(1).translation;  
end-Proc;
```

With data received, we can use DATA-INTO to interpret the JSON, just as the HTTPAPI example did.

(SQL's JSON\_TABLE would've also worked.)

66

## AXIS Example (8 of 9)



```
dcl-proc getHttpStatus;  
  
  dcl-pi *n varchar(10);  
    transportHandle pointer value;  
  end-pi;  
  
  dcl-s result varchar(10) inz('');  
  dcl-s statusCode pointer;  
  
  if transportHandle <> *null;  
    axiscTransportGetProperty( transportHandle  
                              : AXISC_PROPERTY_HTTP_STATUS_CODE  
                              : %addr(statusCode) );  
  
  endif;  
  
  if statusCode <> *null;  
    result = %str(statusCode);  
  endif;  
  
  return result;  
end-proc;
```

axiscTransportGetProperty can be used to get the HTTP status code

200=OK  
403=Forbidden  
404=Not Found  
500=Server-Side Error

67

## AXIS Example (9 of 9)



```
lastCode = axiscTransportGetLastErrorcode(transportHandle);  
lastMsg = %str(axiscTransportGetLastError(transportHandle));  
  
if lastCode = EXC_TRANSPORT_HTTP_EXCEPTION;  
  statusCode = getHttpStatus(transportHandle);  
endif;
```

To save time/space I won't show you the entire error checking routine, just the important parts.

This gets the error number and message.

If the message indicates an HTTP error, it also gets the HTTP status code.

68

## Comparison



### HTTPAPI / YAJS

- Easy to use, performs very well
- Full-featured, offers capabilities that none of the others do
- Requires you to download/install 3<sup>rd</sup> party software

### SQL Functions in QSYS2

- Easy to use, performs very well
- Has most needed features; missing multipart; missing error details
- Included with OS, nothing to install -- but need to be up-to-date

### SQL Functions in SYSTOOLS

- Easy to use, performs poorly (due to Java JVMs)
- Same features as QSYS2, but less than HTTPAPI
- Included with OS, available since 2014

### AXIS C Transport API

- More difficult to use; pointer techniques; more code to do the same things
- Only does HTTP, none of the additional needed features
- Included with OS, nothing to install

69

## This Presentation



You can download a PDF copy of this presentation, as well as other related materials from:

<http://www.scottklement.com/presentations/>

# Thank you!

70