

Web Services for RPGers



Presented by

Scott Klement

<http://www.scottklement.com>

© 2010-2012, Scott Klement

"A computer once beat me at chess, but it was no match for me at kick boxing." — Emo Philips

Our Agenda



This workshop consists of three parts:



1. Introduction
 - What's a web service?
 - Why web services?
 - Terminology and syntax
 - Types
2. Providing a Web Service with RPG
 - Manually, with Apache
 - Using the IBM's Integrated Web Services
3. Consuming a Web Service with RPG
 - With SoapUI / HTTPAPI
 - With WSDL2RPG

I am a Web Service. What Am I?



A routine (program? Subprocedure?) that can be called over a TCP/IP network. (Your LAN? Intranet?)

- A callable routine. (Program? Subprocedure?)
- Callable over a TCP/IP Network. (LAN? Intranet? Internet?)
-can also be called from the same computer.
- Using the HTTP (or HTTPS) network protocol

Despite the name, not necessarily "web"

- different from a "web site" or "web application"
- input and output are via "parameters" (of sorts) and are for programs to use. No user interface -- not even a browser.
- can be used *from* a web application (just as an API or program could) either from JavaScript in the browser, or from a server-side programming language like RPG, PHP, .NET or Java
- but is just as likely to be called from other environments... even 5250!

3

Write Once, Call From Anywhere



In other words... Services Oriented Architecture (SOA).

- Your business logic (business rules) are implemented as a set of "services" to any caller that needs them.
- Web services are only one of many ways to implement SOA. Don't believe the hype!

Callable from anywhere

- Any other program, written in (just about) any language.
- From the same computer, or from another one.
- From the same office (data center), or from another one.
- From folks in the same company, or (if desired) any of your business partners. Even the public, if you want!

RPG can function as either a *provider* (server) or a *consumer* (client)

4

Like Any Other Program Call



A “program call” (or subprocedure call) that works over the Web.

- Very similar in concept to the CALL command.

```
CALL PGM(EXCHRATE) PARM('us' 'euro' &DOLLARS &EUROS)
```

- Runs over the Web, so can call programs on other computers anywhere in the world.

Imagine what you can do....

5

Imagine these scenarios...



Imagine some scenarios:

- You're writing a program that generates price quotes. Your quotes are in US dollars. Your customer is in Germany. You can call a program that's located out on the Internet somewhere to get the current exchange rate for the Euro.
- You're accepting credit cards for payment. After your customer keys a credit card number into your application, you call a program on your bank's computer to get the purchase approved instantly.
- You've accepted an order from a customer, and want to ship the goods via UPS. You can call a program running on UPS's computer system and have it calculate the cost of the shipment while you wait.
- Later, you can track that same shipment by calling a tracking program on UPS's system. You can have up-to-the-minute information about where the package is.

6

More Examples



United Parcel Service (UPS) provides web services for:

- Verifying Package Delivery
 - Viewing the signature that was put on a package
 - Package Time-in-Transit
 - Calculating Rates and Services
 - Obtaining correct shipping information (zip codes, etc.)
-
- FedEx provides web services as well.
 - United States Postal Service
 - Amazon.com
-
- Validate Credit Cards
 - Get Stock Quotes
 - Check the Weather

7

Better Than a Web App?



**Designed to be called from other programs,
instead of interfacing directly with the user.**

- How are they different from web pages?
- Or applications that use HTML for their user interface?
- Can't I just call a regular web app from a program?
- What good is a Web service, then?

***Well, let's start by examining how a simple
web application might work...***

8

Web Applications

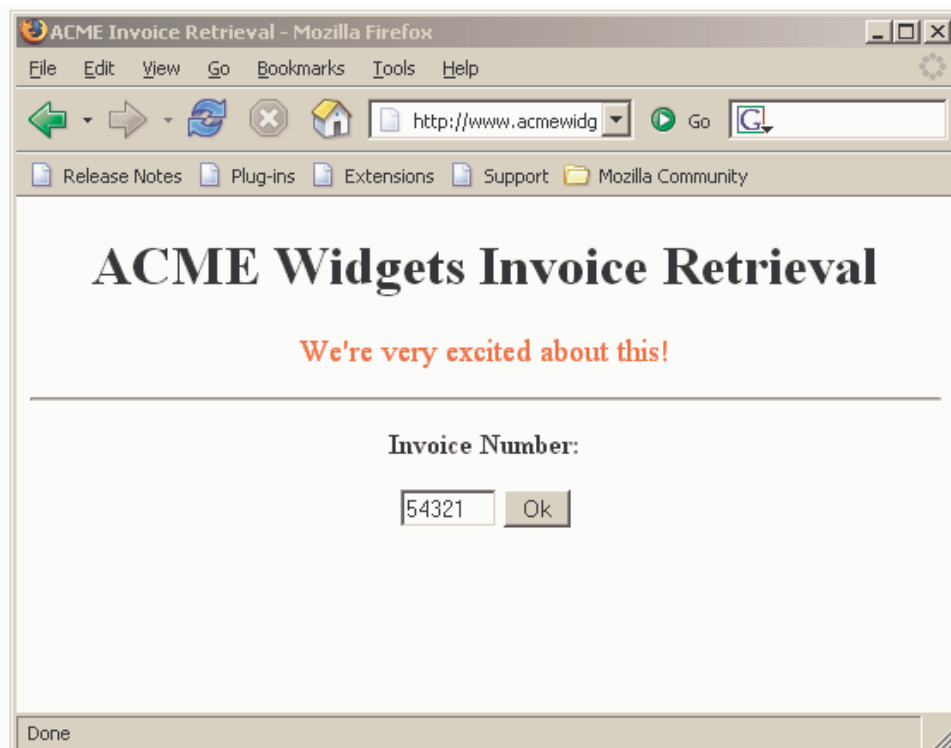


Flow of a typical web application...

- A Web browser displays a web page containing input fields
- The user types some data or makes some selections
- The browser sends the data to a web server which then passes it on to a program
- After processing, the program spits out a new web page for the browser to display

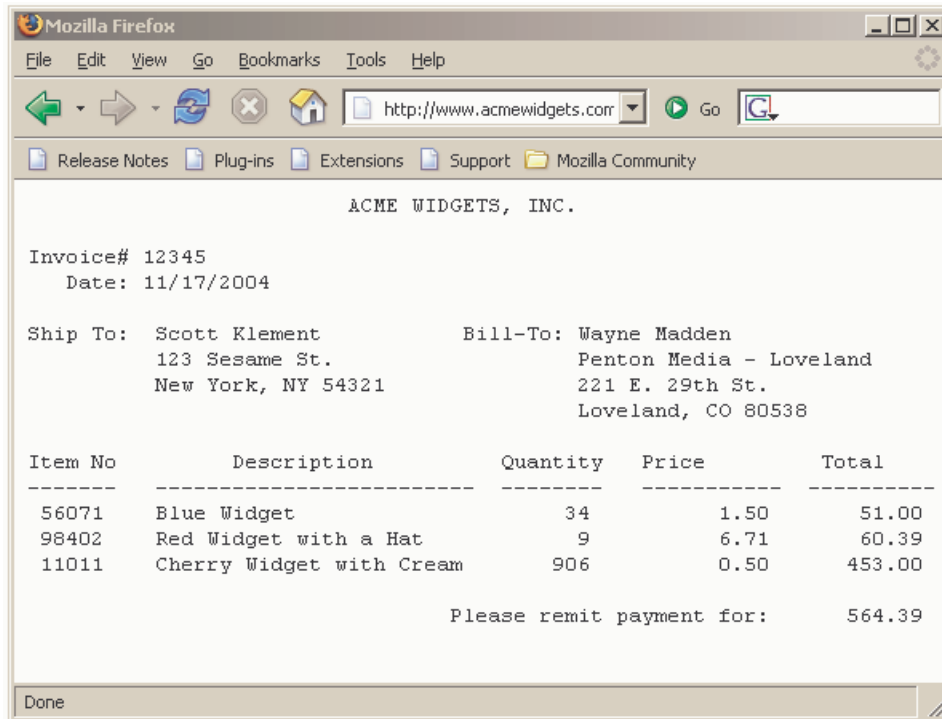
9

Web Enabled Invoice



10

Web Enabled Invoice



11

An idea is born



Eureka! Our company could save time!

- Automatically download the invoice in a program.
- Read the invoice from the download file, get the invoice number as a substring of the 3rd line
- Get the date as a substring of the 4th line
- Get the addresses from lines 6-9

Problem: The data is intended for people to read. Not a computer program!

- Data could be moved, images inserted, colors added
- Every vendor's invoice would be complex & different

12

Need to Know "What"



What you want to know is *what* things are, rather than:

- Where they sit on a page.
- What they look like

The vendor needs to send data that's "marked up."

13

"Marked Up" Data



The screenshot shows a Mozilla Firefox browser window displaying an invoice from ACME WIDGETS, INC. The invoice details are as follows:

Invoice# 12345
Date: 11/17/2004

Ship To: Scott Klement
123 Sesame St.
New York, NY 54321

Bill-To: Wayne Madden
Penton Media - Loveland
221 E. 29th St.
Loveland, CO 80538

Item No	Description	Quantity	Price	Total
56071	Blue Widget	34		51.00
98402	Red Widget with a Hat	9	6.71	60.39
11011	Cherry Widget with Cream	906	0.50	453.00
Please re				564.39

Orange boxes and arrows highlight the following elements:

- Invoice Number**: Points to the invoice number 12345.
- Date**: Points to the date 11/17/2004.
- Ship To**: Points to the ship-to address.
- Bill To**: Points to the bill-to address.
- Total**: Points to the total amount 564.39.
- List of Items**: Points to the item list table.

14

One Way to "Mark Up" is XML



Quick XML Syntax Review

Elements

- An XML opening tag and closing tag.
- Optionally with character data in between.

```
<company> Acme Widgets, Inc </company>
```

(opening) char data (closing)
- Elements can be nested (see next slide)

Attributes

- Looks like a variable assignment

```
<company name="Acme Widgets, Inc"> </company>
```
- Opening/Closing Can Be Combined (a "shortcut")

```
<company name="Acme Widgets, Inc" />
```
- Possible to have multiple attributes and character data

```
<company custno="1234" type="remit">Acme Widgets, Inc</company>
```

15

"Marked Up" Data with XML



```
<invoice>
  <remitto>
    <company>Acme Widgets, Inc</company>
  </remitto>
  <shipto>
    <name>Scott Klement</name>
    <address>
      <addrline1>123 Sesame St.</addrline1>
      <city>New York</city>
      <state>NY</state>
      <postalCode>54321</postalCode>
    </address>
  </shipto>
  <billto>
    <name>Wayne Madden</name>
    <company>Penton Media - Loveland</company>
    <address>
      <addrline1>221 E. 29th St.</addrline1>
      <city>Loveland</city>
      <state>CO</state>
      <postalCode>80538</postalCode>
    </address>
  </billto>
```

16

"Marked Up" Data with XML



```
<itemlist>
  <item>
    <itemno>56071</itemno>
    <description>Blue Widget</description>
    <quantity>34</quantity>
    <price>1.50</price>
    <linetotal>51.00</linetotal>
  </item>
  <item>
    <itemno>98402</itemno>
    <description>Red Widget with a Hat</description>
    <quantity>9</quantity>
    <price>6.71</price>
    <linetotal>60.39</linetotal>
  </item>
  <item>
    <itemno>11011</itemno>
    <description>Cherry Widget</description>
    <quantity>906</quantity>
    <price>0.50</price>
    <linetotal>453.00</linetotal>
  </item>
</itemlist>
<total>564.39</total>
</invoice>
```

17

XML Is Only One Option



Most web services use XML

- As discussed, it identifies "what"
- Possible to add more info without breaking compatibility
- Readable from any modern programming language
- Self-describing (well, sort of.)

Not all web services use XML

- Some do use it for both input and output
- Some use it only for output, and get input via URL
- Some use other formats (most commonly, JSON)

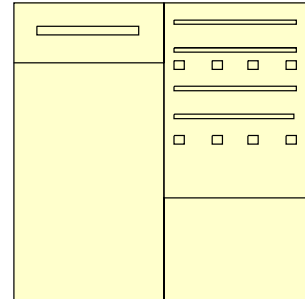
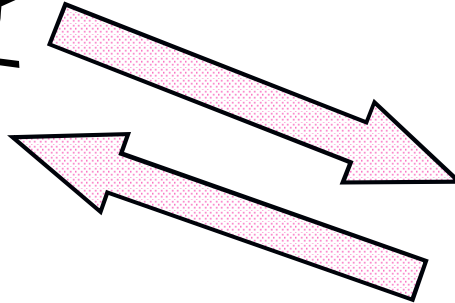
18

How Do They Work?



HTTP starts with a request for the server

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



HTTP then runs server-side program

- input document is given to program
- HTTP waits til program completes.
- program outputs a new document (XML, JSON, etc)
- document contains "output parameters"
- document is returned to calling program.

19

REST Web Services



```
http://www.scottklement.com/cust/495
```

- REST means "REpresentational State Transfer"
- The URL is said to "represent" an object -- and provides the input parameters
 - I like to think of it as "the noun"
- `http` ← the network protocol
- `www.scottklement.com` ← the server
- `/cust/495` ← the thing you want to act upon (the "noun")
- The HTTP "method" (like an opcode) theoretically provides the "verb"
- Due to software limitations, sometimes part of the URL is used for the verb instead of the HTTP method.

Possible methods (and how they "change the state" of the object)

- `GET` (default) -- retrieve the customer -- same as typing URL into browser.
- `POST` -- modify the customer
- `PUT` -- create the customer
- `DELETE` -- delete the customer

20

RESTful Example



Easier way to think of REST

- all input is in URL
- output has no standard... can be anything (but usually is XML or JSON)

For example, you might have a web service that takes a customer number as input, and returns that customer's address.

Input

```
GET http://www.scottklement.com/cust/495
-or-
GET http://www.scottklement.com/cust/495?op=retrieve
```

Output

```
<result>
  <cust id="495">
    <name>ANCO FOODS</name>
    <street>1100 N.W. 33RD STREET</street>
    <city>POMPANO BEACH</city>
    <state>FL</state>
    <postal>33064-2121</postal>
  </cust>
</result>
```

21

REST With Multiple Parameters



- Although the previous slide had only one parameter, REST can have multiple parameters -- but they must all fit on the same URL.

```
http://www.scottklement.com/invoice/495/20100901/20100930
```

- This web service is designed to return a list of invoices for a given customer number, within a given date range.
- 495 = customer number
- 20100901 = start date (in year, month, date format)
- 20100930 = end date (in year, month, date format)

The web service would scan for the slashes, get the parameter info from the URL, and build an XML document that matches the criteria.

Hope you get the idea... Since our time is limited, I *won't* show the XML details for this REST service -- but I *will* implement the same service with POX

22

POX Web Services



<http://www.scottklement.com/poxlib/invoice.pgm>

- POX means "Plain Old XML"
- The URL points to a program that processes the XML
- The input message can be any XML document (you design it!)
- The program reads the XML, gets the input, processes it, and sends output
- Output can be any XML document (you design it!)
- Notice that the preceding URL doesn't contain any input parameters, it merely tells the system which program to run.
- The input parameters are uploaded from an XML file...

23

Input and Output in XML



Input

```
<histQuery>
  <custno>4997</custno>
  <strdate>20100930</strdate>
  <enddate>20100930</enddate>
</histQuery>
```

Output

```
<invoiceList>
  <invoice id="76422">
    <date>20100930</date>
    <name>JACKIE OLSON</name>
    <amount>24.00</amount>
    <weight>8.0</weight>
  </invoice>
  <invoice id="76424">
    <date>20100930</date>
    <name>REYNLDO DE LA TORE</name>
    <amount>5.00</amount>
    <weight>5.0</weight>
  </invoice>
</invoiceList>
```

SOAP Web Services



- Identical to POX, except it follows the SOAP standards for the XML documents.
- Also sends/receives data in XML format. Always XML.
- The XML represents the "parameters"
- Upload an XML document with "input parameters"
- ...then download an XML document with "output parameters"
- An additional "verb" can be supplied in the SoapAction parameter.
- Format of XML is heavily standardized.
- Always accompanied by a WSDL file (though, the other types can be, too)
- XML is designed to be understood/generated by tools.
- By far the most complicated alternative... tooling is almost a requirement.

25

How Would You Tell the World?



So... Web Services are Essentially Program Calls

- But different, because it's over "the web" (http)
- Every web service is different from the next.
- There are different methods of passing "parameters"
 - *and all of those methods are different from traditional RPG parameters!*

If you wrote a reusable program, and wanted everyone to use it, how would you explain it?

- Which server is it on?
- Which network protocol should you call it with?
- What parameters does it accept? (Sequence, data types, etc)

Would you use?

- Documentation in MS Word? Or PDF? Or a wiki somewhere?
- Maybe you'd teach other programmers in person? (like me!)
- Comments in the code?
- Expect the caller to read the code??!

26

WSDL Files



Web Services Description Language (WSDL)

- pronounced "WHIZ-dull"
- Standardized way of documenting a web service.
- A type (schema? flavor?) of XML
- Can be generated by a tool from your parameter list!
- Can be read by a computer program to make your service easy to call
- Almost always used with SOAP. Occasionally also used with POX or REST.

Describes the web service:

- What it does
- What routines it offers (like procedures in a service program)
- Where the service is located (domain name or IP address)
- Protocol to use
- Structure of input/output messages (parameters)

27

WSDL Skeleton



```
<definitions>
  <types>
    definition of types.....
  </types>
  <message>
    definition of a message....
  </message>
  <portType>
    definition of a port.....
  </portType>
  <binding>
    definition of a binding....
  </binding>
  <service>
    a logical grouping of ports...
  </service>
</definitions>
```

<types> = the data types that the web service uses.
<message> = the messages that are sent to and received from the web service.
<portType> = the operations (or, "programs/procedures" you can call for this web service.
<binding> = the network protocol used.
<service> = a grouping of ports. (Much like a service program contains a group of subprocedures.)

28

SOAP



SOAP = Simple Object Access Protocol

SOAP is an XML language that describes the parameters that you pass to the programs that you call. When calling a Web service, there are two SOAP documents -- an input document that you send to the program you're calling, and an output document that gets sent back to you.

"Simple" is a relative term!

- Not as simple as RPG parameter lists.
- Not as simple as REST or POX
- Simpler than CORBA.

- Pretty much identical to POX, except the XML is standardized -- and there's a WSDL!

29

SOAP Skeleton



Here's the skeleton of a SOAP message:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding" >

  <soap:Header>
    (optional) contains header info, like payment info or authentication info
    (crypto key, userid/password, etc)
  </soap:Header>

  <soap:Body>
    . . .
    Contains the parameter info. (Varies by application.)
    . . .
    <soap:Fault>
      (optional) error info.
    </soap:Fault>
    . . .
  </soap:Body>

</soap:Envelope>
```

30

Namespaces



SOAP always uses name spaces

- you combine your parameter data (user defined XML) with SOAP XML
- chance of conflicting names!
- name spaces keep them unique
- the URI of a name space isn't connected to over the network, it just guarantees uniqueness (*there's only one w3.org!*)

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  xmlns:morgue="http://example.morgue.com/xml/cadavernet">
  <soap:Body>
    <morgue:CadaverArray>
      <morgue:Body>
        <morgue:lastName>Klement</morgue:lastName>
        <morgue:firstName>Scott</morgue:firstName>
      </morgue:Body>
      <morgue:Body>
        <morgue:lastName>Smith</morgue:lastName>
        <morgue:firstName>Paul</morgue:firstName>
      </morgue:Body>
    </morgue:CadaverArray>
  </soap:Body>
</soap:Envelope>
```

31

More Namespace Notes



- An xmlns without a prefix designates the "default" namespace.
- It's the URI, not the prefix that identifies the namespace.
(in the example below, tns:Body and Body are interchangeable)
- Until recently, XML-INTO had very poor support for name spaces. (A recent PTF added better namespace capability for 6.1+)

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  xmlns:tns="http://example.morgue.com/xml/cadavernet"
  xmlns="http://example.morgue.com/xml/cadavernet" >
  <soap:Body>
    <CadaverArray>
      <tns:Body>
        <lastName>Klement</lastName>
        <firstName>Scott</firstName>
      </tns:Body>
      <Body>
        <lastName>Smith</lastName>
        <firstName>Paul</firstName>
      </Body>
    </CadaverArray>
  </soap:Body>
</soap:Envelope>
```

Currency Exchange Example



- Free "demo" web service from WebServiceX.net
- The most frequently used sample that's included with HTTPAPI

If you've never used it before, how would you find it?

- Browsing a site like WebServiceX.net
- Or XMethods.net
- Or BindingPoint.com
- Or RemoteMethods.com
- Or simply Google for "(SUBJECT) WSDL"
 - such as "Currency Exchange WSDL"
- Download the WSDL file to learn about the service.
- Almost everyone will use a tool (software) to understand WSDL
- I prefer an open source tool called SoapUI (which is available in both a "free" and "for money/supported" version.)

The WSDL will (of course) tell you what the SOAP messages would look like

33

Sample SOAP Documents



I've removed the namespace information to keep this example clear and simple. (In a real program, you'd need those to be included as well.)

Input Message

```
<?xml version="1.0"?>
<SOAP:Envelope (namespaces here)>
  <SOAP:Body>
    <ConversionRate>
      <FromCurrency>USD</FromCurrency>
      <ToCurrency>EUR</ToCurrency>
    </ConversionRate>
  </SOAP:Body>
</SOAP:Envelope>
```

Output Message

```
<?xml version="1.0"?>
<SOAP:Envelope (namespaces here)>
  <SOAP:Body>
    <ConversionRateResponse>
      <ConversionRateResult>0.7207</ConversionRateResult>
    </ConversionRateResponse>
  </SOAP:Body>
</SOAP:Envelope>
```

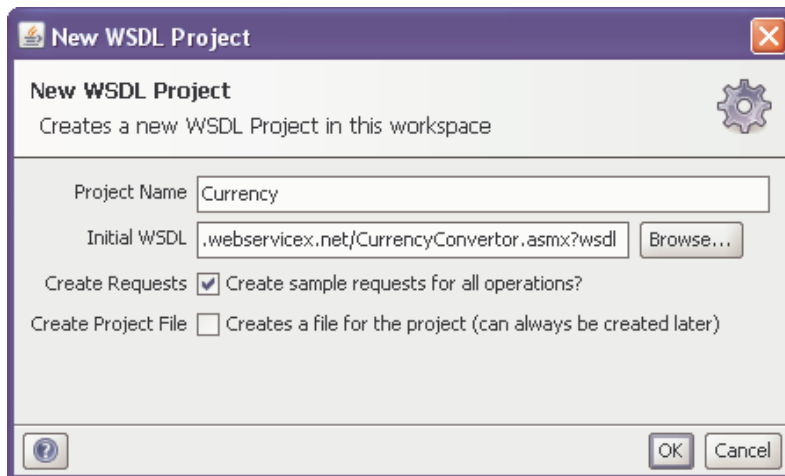
34

SoapUI (1/2)



SoapUI is an open source (free of charge) program that you can use to get the SOAP messages you'll need from a WSDL document. <http://www.soapui.org>

Click **File / New WSDL Project**

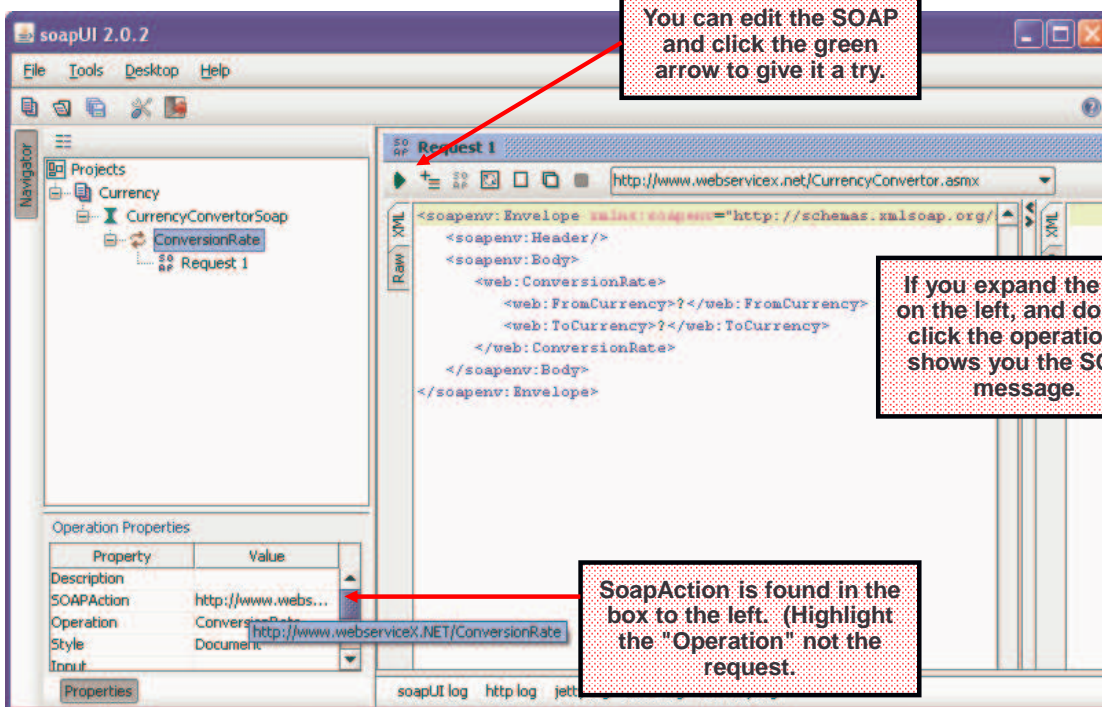


PROJECT NAME
can be any name – use something you'll remember.

INITIAL WSDL
can be either a URL on the web, or a file on your hard drive.

You can use "Browse" to navigate via a standard Windows file dialog.

SoapUI (2/2)



You can edit the SOAP and click the green arrow to give it a try.

If you expand the tree on the left, and double-click the operation, it shows you the SOAP message.

SoapAction is found in the box to the left. (Highlight the "Operation" not the request.

Exercises



- Look around the Internet for some REST web services. Try them in your browser, see what you get with different input values.
- Try the same thing with POX services. Just type an XML document into an editor, and send it with an HTTP tool like HTTPAPI.
- Install SoapUI on your PC.
- Try loading different WSDLs into SoapUI, changing the parameters, and running them.

37

RPG as a Web Service Provider



Presented by

Scott Klement

<http://www.scottklement.com>

© 2010-2011, Scott Klement

"If you give someone a program, you will frustrate them for a day; if you teach them how to program, you will frustrate them for a lifetime."

Break is Over, time for REST



To get started with REST, let's tell Apache how to call our program.

```
ScriptAlias /cust /qsys.lib/restful.lib/custinfo.pgm
<Directory /qsys.lib/restful.lib>
  Order Allow,Deny
  Allow From All
</Directory>
```

- Just add the preceding code to an already working Apache instance on IBM i.
- `ScriptAlias` tells apache that you want to run a program.
- If URL starts with `/cust`, Apache will **CALL PGM(RESTFUL/CUSTINFO)**
- Our REST web service can be run from any IP address (Allow from all).

```
http://as400.klements.com/cust/495
```

- Browser connects to: `as400.klements.com`
- Apache sees the `/cust` and calls `RESTFUL/CUSTINFO`
- Our program can read the 495 (customer number) from the URL itself.

39

This is CGI -- But It's Not HTML



Web servers (HTTP servers) have a standard way of calling a program on the local system. It's know as Common Gateway Interface (CGI)

- The URL you were called from is available via the `REQUEST_URI` env. var
- If any data is uploaded to your program (not usually done with REST) you can retrieve it from "standard input".
- To write data back from your program to Apache (and ultimately the web service consumer) you write your data to "standard output"

To accomplish this, I'm going to use 3 different APIs (all provided by IBM)

- `QtmhRdStin` ← reads standard input
- `getenv` ← retrieves an environment variable.
- `QtmhWrStout` ← writes data to standard output.

40

Example REST Provider (1 of 3)



FCUSTFILE	IF	E	K	DISK
D getenv		PR	*	extproc('getenv')
D var			*	value options(*string)
D QtmhWrStout		PR		extproc('QtmhWrStout')
D DtaVar			65535a	options(*varsize)
D DtaVarLen			10I 0	const
D ErrorCode			8000A	options(*varsize)
D err		ds		qualified
D bytesProv			10i 0	inz(0)
D bytesAvail			10i 0	inz(0)
D xml		pr	5000a	varying
D inp			5000a	varying const
D CRLF		C		x'0d25'
D pos		s	10i 0	
D uri		s	5000a	varying
D data		s	5000a	

41

Example REST Provider (2 of 3)



```
/free
uri = %str( getenv('REQUEST_URI') );
monitor;
pos = %scan('/cust/': uri) + %len('/cust/');
custno = %int(%subst(uri:pos));
on-error;
data = 'Status: 500 Invalid URI' + CRLF
      + 'Content-type: text/xml' + CRLF
      + CRLF
      + '<error>Invalid URI</error>' + CRLF;
QtmhWrStout(data: %len(%trimr(data)): err);
return;
endmon;

chain custno CUSTFILE;
if not %found;
data = 'Status: 500 Unknown Customer' + CRLF
      + 'Content-type: text/xml' + CRLF
      + CRLF
      + '<error>Unknown Customer Number</error>' + CRLF;
QtmhWrStout(data: %len(%trimr(data)): err);
return;
endif;
```

REQUEST_URI will contain
http://x.com/cust/495

Custno is everything
after /cust/ in the URL

If an error occurs, I set
the status to 500, so the
consumer knows there
was an error. We also
provide a message in
XML, in case the
consumer wants to
show the user.

Example REST Provider (3 of 3)



```
data = 'Status: 200 OK' + CRLF
+ 'Content-type: text/xml' + CRLF
+ CRLF
+ '<result>'
+ '<cust id="' + %char(custno) + '"'
+ '<name>' + xml(name) + '</name>'
+ '<street>' + xml(street) + '</street>'
+ '<city>' + xml(city) + '</city>'
+ '<state>' + xml(state) + '</state>'
+ '<postal>' + xml(postal) + '</postal>'
+ '</cust>'
+ '</result>' + CRLF;

QtmhWrStout(data: %len(%trimr(data)): err);
```

Status 200 means that all was well.

Here I send the XML Response.

The xml() subprocedure is just a little tool to escape any special characters that might be in the database fields.

I won't include the code for that in this talk, but you can download the complete program from my web site (see link at end of handout.)

43

Test the REST



The screenshot shows a Mozilla Firefox browser window with the address bar set to `http://as400.klements.com/cust/495`. The page content displays the following XML response:

```
- <result>
- <cust id="495">
  <name>ANCO FOODS </name>
  <street>1100 N.W. 33RD STREET </street>
  <city>POMPANO BEACH </city>
  <state>FL</state>
  <postal>33064-2121</postal>
</cust>
</result>
```

You can test REST web services that use the GET method simply by typing the URL into your browser.

44

Test REST with HTTPAPI



This requires HTTPAPI version 1.24 or later

```
HTTPAPI REQTYPE(*GET)
        URL('http://as400.klements.com/cust/520')
        DEBUG('/tmp/restdebug.txt')
        DOWNLOAD('/tmp/resttest.xml')
```

If successful, the output from the web service will now be in the DOWNLOAD file, shown above as `/tmp/resttest.xml`. Use the IBM i DSPF command to view it on green-screen:

```
DSPF STMF('/tmp/resttest.xml')
```

HTTPAPI supports both GET and POST methods, and PUT & DELETE are planned for the future. Plus, the DEBUG file can provide extra debugging information when something is wrong.

```
DSPF STMF('/tmp/restdebug.xml')
```

45

A POX on Your REST!



Advantages of REST server

- Relatively simple to code with RPG/Apache.
- Simple enough that no special tools are needed, it's reasonable for you to test/develop it yourself.
- Runs very efficiently.
- Easy to test (just use a browser.)
- Especially useful when you'll be implementing both the client/server inhouse.
- Very easy to call from JavaScript (AJAX)

Disadvantages of REST server

- Input parameters are very limited
- Input must be URL encoded, and legal in a URL
- Input length is limited (approx 2000 total length -- varies from browser to browser)
- No tooling supports it.
- There's no standard to the format of input/output information

Some of these limitations (mainly the input parameter limitations) can be solved by using POX.

46

Apache Meets POX



To get started with REST, let's tell Apache how to call our program.

```
ScriptAlias /poxlib /qsys.lib/poxlib.lib
<Directory /qsys.lib/poxlib.lib>
  Order Allow,Deny
  Allow From All
</Directory>
```

- Just add the preceding code to an already working Apache instance on IBM i.
- `ScriptAlias` tells apache that you want to run a program.
- If URL starts with `/poxlib`, Apache will call pgms in POXLIB library.
- Our POX web services can be run from any IP address.

```
http://as400.klements.com/poxlib/invoice.pgm
```

- Browser connects to: `as400.klements.com`
- Apache sees the `/poxlib/invoice.pgm` and calls POXLIB/INVOICE
- Both input & output parameters are in XML

47

Example POX Provider (1 of 4)



```
D QtmhRdStin      PR                extproc('QtmhRdStin')
D   RcvVar        65535a      options(*varsize)
D   RcvVarLen     10I 0      const
D   LenAvail      10I 0
D   ErrorCode     8000A      options(*varsize)

D QtmhWrStout     PR                extproc('QtmhWrStout')
D   DtaVar        65535a      options(*varsize) const
D   DtaVarLen     10I 0      const
D   ErrorCode     8000A      options(*varsize)

D err             ds                qualified
D   bytesProv     10i 0      inz(0)
D   bytesAvail    10i 0      inz(0)

D xml             pr                5000a      varying
D   inp           5000a      varying const

D CRLF           C                x'0d25'
D inputLen       s                10i 0
D inputXml       s                65535a
D data           s                5000a
```

48

Example POX Provider (2 of 4)



```
// inputXml will look like this:
//
//      <histQuery>
//          <custno>4997</custno>
//          <strdate>20100901</strdate>
//          <enddate>20100930</enddate>
//      </histQuery>

D inputParm      ds          qualified
D  custno        4s 0
D  strdate       8s 0
D  enddate       8s 0

/free
inputXml = *blanks;

QtmhRdStin( inputXml
            : %size(inputXml)
            : inputLen
            : err );

xml-into inputParm %xml(inputXml: 'path=histQuery');
```

With XML-INTO, the DS subfields need to match the XML element or attribute names.

QtmhRdStin() loads the XML document uploaded from the consumer into a variable named inputXml

xml-into parses the XML, and loads it into the inputParm DS.

49

Example POX Provider (3 of 4)



```
exec SQL declare C1 cursor for
select aiOrdn, aiIDat, aiSNme, aiDamt, aiLbs
from ARSHIST
where aiCust = :inputParm.custno
and aiIDat between :inputParm.strdate
and :inputParm.enddate;

exec SQL open C1;
exec SQL fetch next from C1 into :row;

if sqlstt <> '00000'
and %subst(sqlstt:1:2) <> '01'
and %subst(sqlstt:1:2) <> '02';
data = 'Status: 500 Query Failed' + CRLF
+ 'Content-type: text/xml' + CRLF
+ CRLF
+ '<error>Failed with SqlState=' + sqlstt + '</error>'
+ CRLF;
QtmhWrStout(data: %len(%trimr(data)): err);
return;
endif;
```

The input parameters are used to look up the invoices for the customer with SQL.

If there are any errors, I use status 500, just as I did in the REST example.

50

Example POX Provider (4 of 4)



```
data = 'Status: 200 OK' + CRLF
      + 'Content-type: text/xml' + CRLF
      + CRLF
      + '<invoiceList>';
QtmhWrStout(data: %len(%trimr(data)): err);

dow sqlstt='00000' or %subst(sqlstt:1:2)='01';
  data = '<invoice id="' + row.inv          + '"' + '>'
        + '<date>'          + %editc(row.date:'X') + '</date>'
        + '<name>'          + xml(row.name)         + '</name>'
        + '<amount>'        + %char(row.amount)    + '</amount>'
        + '<weight>'        + %char(row.weight)    + '</weight>'
        + '</invoice>';
  QtmhWrStout(data: %len(%trimr(data)): err);
  exec SQL fetch next from C1 into :row;
enddo;

exec SQL close C1;

data = '</invoiceList>' + CRLF;
QtmhWrStout(data: %len(%trimr(data)): err);
```

The writes to standard output are appended to each other. (Much like adding records to a file.)

So I can write XML data in a loop, and the result will be one big XML document to the consumer.

51

Testing for POX (1 of 2)



To test POX, we need to create an XML file to upload. The file must be ASCII.

```
QSH CMD('touch -C 819 /tmp/poxinput.xml')
EDTF STMF('/tmp/poxinput.xml')
```

Enter this:

```
<histQuery>
  <custno>4997</custno>
  <strdate>20100901</strdate>
  <enddate>20100930</enddate>
</histQuery>
```

Now send it to the POX web service (*again, requires HTTPAPI 1.24+*)

```
HTTPAPI REQTYPE(*POST)
  URL('http://as400.klements.com/poxlib/invoice.pgm')
  UPLOAD('/tmp/poxinput.xml')
  DOWNLOAD('/tmp/poxoutput.xml')
  DEBUG('/tmp/poxdebug.txt')
```

Testing for POX (2 of 2)



To test POX, we need to create an XML file to upload. The file must be ASCII.

```
DSPF STMF (' /tmp/poxoutput.xml ')
```

```
Browse : /tmp/poxoutput.xml
Record : 1 of 220 by 14          Column : 1      80 by 79
Control :

.....1.....2.....3.....4.....5.....6.....7.....
*****Beginning of data*****
<invoiceList><invoice id="70689"><date>20100901</date><name>JIM JOHNSON
  </name><amount>14.80</amount><weight>3.5</weight></invoice><invoice id="706
95"><date>20100901</date><name>BILL VIERS          </name><amount>9.80</amo
unt><weight>3.2</weight></invoice><invoice id="70700"><date>20100901</date><name
>JOSE MENDOZA          </name><amount>6.00</amount><weight>3.0</weight></invo
ice><invoice id="70703"><date>20100901</date><name>RICHARD KERBEL          </na
me><amount>10.00</amount><weight>5.0</weight></invoice><invoice id="70715"><date
>20100901</date><name>JACKIE OLSON          </name><amount>23.80</amount><wei
ght>10.0</weight></invoice><invoice id="70736"><date>20100901</date><name>LISA X
IONG          </name><amount>24.00</amount><weight>7.0</weight></invoice><i
nvoice id="70748"><date>20100901</date><name>JOHN HANSON          </name><am
ount>11.80</amount><weight>5.0</weight></invoice><invoice id="70806"><date>20100
901</date><name>JOHN ESSLINGER          </name><amount>7.50</amount><weight>5.0
</weight></invoice><invoice id="70809"><date>20100901</date><name>LORI SKUZENSKI

F3=Exit   F10=Display Hex   F12=Exit   F15=Services   F16=Repeat find
F19=Left   F20=Right
Record length of 80 used.
```

Tip: If you download the XML file to your PC, and view it with a browser, it'll be much easier to read!

Clean up the POX with SOAP



Advantages to POX

- Still relatively simple.
- Doesn't have the limitations on input parameters that REST has.

Disadvantages of POX

- Still no standard
- Still no tooling
- Slower and more complicated than REST.
- How would you document it for other programmers to use? If you published it to the whole world, how easy would it be for them to implement it?

SOAP, with WSDL, is more complicated yet. But its format is standardized, and the tooling available for SOAP makes it easier for the "average joe on the street" to call your web service.

IBM's Integrated Web Services Server



We could do SOAP the same way as POX -- but we'd have to develop the WSDL file manually, and that would be difficult. Fortunately, IBM provides a Web Services tool with IBM i at no extra charge!

The tool takes care of all of the HTTP and XML work for you!

It's called the *Integrated Web Services* tool.

<http://www.ibm.com/systems/i/software/iws/>

- Can be used to provide web services
- Can also be used to consume them -- but requires in-depth knowledge of C and pointers -- I won't cover IBM's consumer tool today.

Requirements:

- IBM i operating system, version 5.4 or newer.
- 57xx-SS1, opt 30: QShell
- 57xx-SS1, opt 33: PASE
- 57xx-JV1, opt 8: J2SE 5.0 32-bit (Java)
- 57xx-DG1 -- the HTTP server (powered by Apache)

Make sure you have the latest cum & group PTFs installed.

55

SOAP Example and PCML



This is an example of creating a SOAP web service that works like the 'CUST' REST example that I showed you previously. To get started, I need an RPG program that uses input/output parameters for the fields we want to return.

PCML = Program Call Markup Language

- A flavor of XML that describes a program's (or *SRVPGM's) parameters.
- Can be generated for you by the RPG compiler, and stored in the IFS:

```
CRTBNDRPG PGM(xyz) SRCFILE(QRPGLESRC)
          PGMINFO(*PCML)
          INFOSTMF('/path/to/myfile.pcml')
```

- Or can be embedded into the module/program objects themselves, with an H-spec:

```
H PGMINFO(*PCML:*MODULE)
```

56

SOAP GETCUST (1 of 2)



```
H DFTACTGRP(*NO) ACTGRP('SOAP') PGMINFO(*PCML: *MODULE)
FCUSTFILE IF E K DISK PREFIX('CUST.')
D CUST E DS qualified
D D extname(CUSTFILE)
D GETCUST PR ExtPgm('GETCUST')
D CustNo like(Cust.Custno)
D Name like(Cust.Name)
D Street like(Cust.Street)
D City like(Cust.City)
D State like(Cust.State)
D Postal like(Cust.Postal)
D GETCUST PI
D CustNo like(Cust.Custno)
D Name like(Cust.Name)
D Street like(Cust.Street)
D City like(Cust.City)
D State like(Cust.State)
D Postal like(Cust.Postal)
```

PCML with parameter info will be embedded in the module and program objects.

This PREFIX causes the file to be read into the CUST data struct.

When there's no P-spec, the PR/PI acts the same as *ENTRY PLIST.

57

SOAP GETCUST (2 of 2)



```
/free
chain CustNo CUSTFILE;
if not %found;
  msgdta = 'Customer not found.';
  QMHSNDPM( 'CPF9897': 'QCPFMSG *LIBL'
           : msgdta: %len(msgdta): '*ESCAPE'
           : '*PGMBDY': 1: MsgKey: err );
else;
  Custno = Cust.Custno;
  Name = Cust.name;
  Street = Cust.Street;
  City = Cust.City;
  State = Cust.State;
  Postal = Cust.Postal;
endif;
*inlr = *on;
/end-free
```

This API is equivalent to the CL SNDPGMMSG command, and causes my program to end with an exception ("halt")

When there are no errors, I simply return my output via the parameter list. IWS takes care of the XML for me!

58

The Wizarding World



Step 1 was to make my RPG program, using parameters to return the result.

Step 2 is to set it up in the IWS's web service wizard.

- It'll do all the XML and HTTP stuff for me.
- Theoretically, I don't need to know anything about the XML...
- In other languages (Java, .NET, PHP, etc) many programmers know nothing about how SOAP web services work internally -- they just know it's a way to call a routine over a network. They're used to the XML being handled for them.

Step 3 will be to test my new web service with SoapUI

59

The IWS Wizard (1 of 13)



To get started, point your browser at:
<http://yoursystem:2001>

Click here to get to the web services wizard.

IBM
(C) IBM Corporation 2000

i5/OS Tasks

AS400 KLEMENTS.COM

- [IBM Web Administration for i5/OS](#)
Configure HTTP servers, application servers and deploy applications
- [iSeries Navigator URL Advisor](#)
Learn how to add i5/OS administration tasks into your web applications
- [Digital Certificate Manager](#)
Create, distribute, and manage Digital Certificates
- [IBM Directory Server for i5/OS](#)
Administer the IBM Directory Server
- [IBM IPP Server for i5/OS](#)
Configure the IBM IPP Server
- [Cryptographic Coprocessor](#)
Configure the cryptographic coprocessor
- [i5/OS Web-Based Help Server](#)
Administer the Web-based help server

60

The IWS Wizard (2 of 13)



Make sure you are on the "manage" tab.

And click "all servers"

Then "Create Web Services Server"

Server	Version	Status	Address:Port	Associated Application Server
ADMIN	Apache/2.0.63	Running	*.2001	None
APACHEDFT	Apache/2.0.63	Stopped	*.80	None
HABRISDATA	Apache/2.0.63	Stopped	192.168.5.4.8082	None
IWADEF	Apache/2.0.63	Stopped	*.2020	None
LANSA	Apache/2.0.63	Stopped	*.8080	None
MAIN	Apache/2.0.63	Running	*.443	None
OIPPSVR	Apache/2.0.63	Running	*.851	None
TEST1	Apache/2.0.63	Running	*.9080	None
ZENDCORE	Apache/2.0.63	Running	*.89	None

The IWS Wizard (3 of 13)



Here you enter the userid that the IBM server (not your RPG program) runs under.

A single server can be used to host many RPG or Cobol programs.

I recommend either taking the IBM default (first option on left) or creating a userid that's only used for this.

Specify user ID for this server:

- Use default user ID
- Specify an existing user ID
- Create a new user ID

User ID:

The IWS Wizard (4 of 13)



HTTP Server Administration on AS400 - Mozilla Firefox

http://as400.klements.com:2001/HTTPAdmin

IBM Web Administration for i

Setup **Manage** Advanced | Related Links

All Servers HTTP Servers | Application Servers | ASF Tomcat Servers

Common Tasks and Wizards

- Create Web Services Server
- Create HTTP Server
- Create Application Server
- Migrate Original to Apache
- Create WebSphere Portal
- Create IBM Workplace

Create Web Services Server

Externalize an IBM i Program as a Web Service - Step 2 of 8

You may externalize an IBM i program object as a Web service. The program object must be an existing Integrated Language Environment (ILE) program (*PGM) or service program (*SRVPGM) object. Currently, only program objects written using the COBOL or RPG programming languages are supported.

Externalize an IBM i program as a Web service:

Deploy new service to externalize an IBM i program.

Click next...

0.5

The IWS Wizard (5 of 13)



IBM Web Administration for i

Setup **Manage** Advanced | Related Links

All Servers HTTP Servers | Application Servers | ASF Tomcat Servers

Common Tasks and Wizards

- Create Web Services Server
- Create HTTP Server
- Create Application Server
- Migrate Original to Apache
- Create WebSphere Portal
- Create IBM Workplace

Create Web Services Server

Deploy New Service: Specify Location of IBM i Program Object - Step 3 of 8

The IBM i object to be externalized as a Web service must be an existing ILE program (*PGM) or service program (*SRVPGM) located on the system. Currently, only program objects written using the COBOL or RPG programming languages are supported.

Specify the library and program object for the Web service.

Specify IBM i library and ILE program object name (Recommended)

You can specify the program object location by entering the name of the library and the name of the program object. This is the fastest way to specify the program object.

Library name:

ILE Object name:

ILE Object type: *SRVPGM *PGM

Browse the integrated file system for the IBM i program object

Back Next Cancel

This is how the IWS finds your program. It will extract parameter info from the embedded PCML.

If you use an external PCML, an extra screen will prompt you for the IFS location.

The IWS Wizard (6 of 13)



IBM Web Administration for i
Setup Manage Advanced | Related Links
All Servers HTTP Servers | Application Servers | ASF Tomcat Servers

Common Tasks and Wizards
Create Web Services Server
Create HTTP Server
Create Application Server
Migrate Original to Apache
Create WebSphere Portal
Create IBM Workplace

Create Web Services Server
Deploy New Service: Specify Name for Service - Step 4 of 8

Specify a unique name for this service. ?

Service name: GETCUST
Service description: Retrieve Customer Address

This is the name & description of your service, as it'll appear in the WSDL, as well as the IWS wizard screens.

The IWS Wizard (7 of 13)



procedure is a set of self-contained high-level language statements that performs a particular task and then returns to the caller. A service program contains one or more procedures. A program contains only one procedure.

The table below lists all the exported procedures through this Web service. Expand the procedure row to view the parameters. The Usage parameter attribute affects the output of the Web service. For array type parameters, modifying the Usage parameter affects the output of the Web service.

Export procedures: ?

Select	Procedure name/Parameter name	Usage	Data type	Count
<input checked="" type="checkbox"/>	GETCUST			
<input type="checkbox"/>	CUSTNO	input/output	zoned	
<input type="checkbox"/>	NAME	output	char	
<input type="checkbox"/>	STREET	output	char	
<input type="checkbox"/>	CITY	output	char	
<input type="checkbox"/>	STATE	output	char	
<input type="checkbox"/>	POSTAL	output	char	

Select All Deselect All Expand All Collapse All
input
input/output
output

Back Next Cancel

Here you specify which parameters should appear in the input SOAP (XML) message, the output SOAP (XML) message or both.

The IWS Wizard (8 of 13)



Now you specify the userid that your RPG program will run under

You can choose "Use server's" if you want the same one selected in step 3.

The IWS Wizard (9 of 13)



Here you configure the exact library list that you want your RPG program to run under.

Use the Add/Remove/Up/Down buttons to add/remove libraries, and change their order.

The IWS Wizard (10 of 13)



IBM Web Administration for i
Setup **Manage** Advanced | Related Links
All Servers | HTTP Servers | Application Servers | ASF Tomcat Servers

Common Tasks and Wizards
Create Web Services Server
Create HTTP Server
Create Application Server
Migrate Original to Apache
Create WebSphere Portal
Create IBM Workplace

Servers Services Operations

Web Services Server Information
Server name: WSERVICE
Server description: Web services server WSERVICE, created by the Create Web Services Server wizard.
Internal port range: 10000 - 10009
Server root: /www/WSERVICE
Server URL: http://as400.klements.co
User ID for server: KLEMSCOT
Context root: /web

HTTP Server Information
HTTP server name: WSERVICE
HTTP server description: HTTP server created by the Create Web Service Server wizard.
Port: 10010
Document root: /www/WSERVICE/htdocs
Server root: /www/WSERVICE
Server association: WSERVICE

Back Finish **Cancel**

The wizard is done. It has all of the information it needs! It displays this summary screen, so you can double-check the settings.

When you click 'Finish', it'll generate the appropriate Java programs to handle your XML and call your RPG program.

The IWS Wizard (11 of 13)



IBM Web Administration for i
Setup **Manage** Advanced | Related Links
All Servers | HTTP Servers | Application Servers | ASF Tomcat Servers

Creating Server: WSERVICE - V1.3 (web services)

Common Tasks and Wizards
Create Web Services Server
Create HTTP Server
Create Application Server
Migrate Original to Apache
Create WebSphere Portal
Create IBM Workplace

Server: WSERVICE
Web services server WSERVICE, created by the Create Web Services Server wizard.

The Web services server is in the process of generating the Java programs for handling XML and calling your RPG program. Web services are based services from standard communication implemented using C, C++, Java and .NET. Web services server and the services for IBM i program objects. Other management functions such as starting, stopping and deleting services are also provided. For more information, please visit: <http://www-03.ibm.com/systems/i/software/iws/>

Server "WSERVICE" is in the process of being created. To update the status, click the Refresh icon above.

Note: To update the status, click Refresh

While it's generating the Java code to call your service, you'll see this screen. Click the "refresh" button every minute or so to check the status.

The IWS Wizard (12 of 13)



The page will contain this box when the process is complete.

Server: "WSERVICE"

- ConvertTemp
- GETCUST

IBM automatically creates the "ConvertTemp" service as a sample to let you test your server.

GETCUST is mine -- the one I just created.

You can do tests, add/delete services, and get the WSDL for your service by clicking the "Manage Deployed Services" link on the left.

When you see this box (with the "green light" balls) it means your service is active!

The IWS Wizard (13 of 13)



IBM Web Administration for i

Setup **Manage** Advanced | Related Links

All Servers | HTTP Servers **Application Servers** ASF Tomcat Servers

Running Server: WSERVICE - V1.3 (web services)

WSERVICE > Manage Deployed Services

Manage Deployed Services

Data current as of Oct 9, 2010 6:30:19 PM.

Deployed services:

Service name	Status	Startup type	WSDL - service definition
ConvertTemp	Running	Automatic	View definition
GETCUST	Running	Automatic	View definition

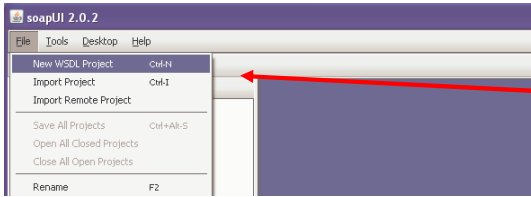
Deploy Stop Properties Uninstall Refresh Test Service

This is what you see when you click "Manage Deployed Services"

Some of the buttons (stop, properties, etc) will only show up after you check the radio button on the left.

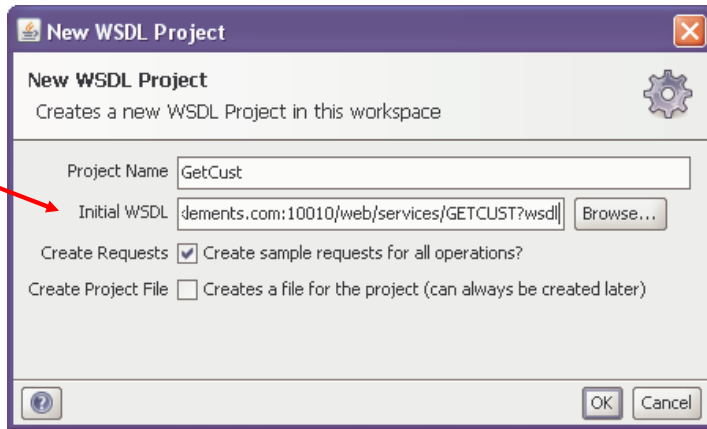
The "View Definition" buttons are links to the WSDL for the corresponding service.

Testing SOAP with SoapUI (1 of 4)

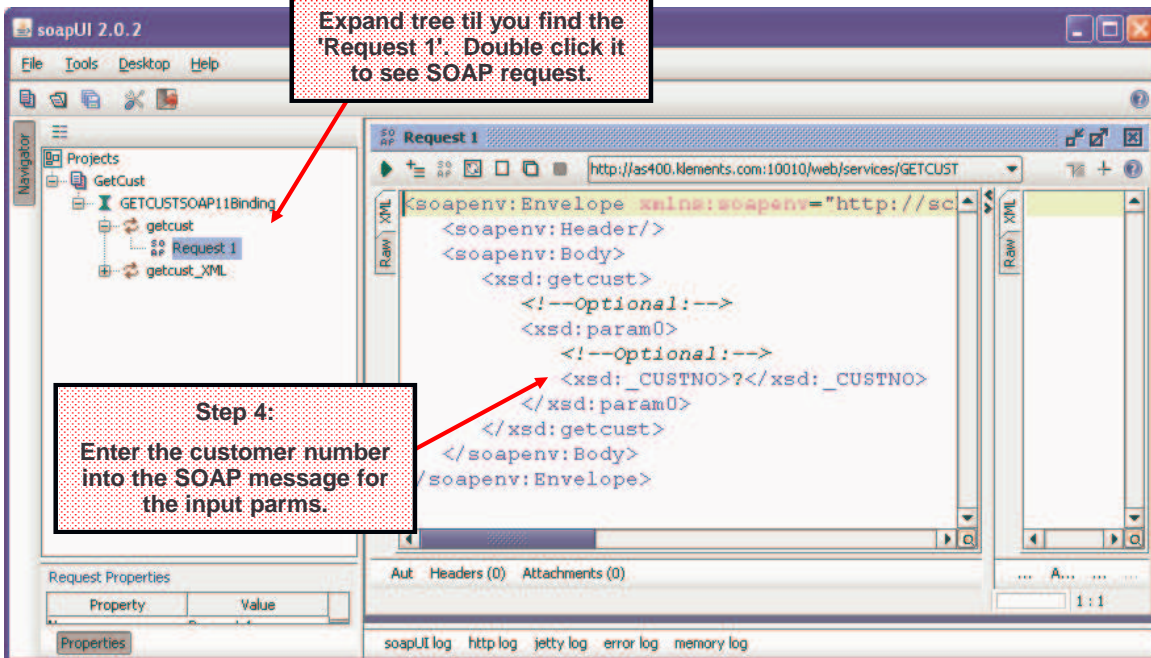


Step 1:
Click File -> New Project
(some versions say "WSDL project", others say "SoapUI project. They're the same.)

Step 2:
Paste in URL to WSDL
(from the "View Service Definition" link into the Initial WSDL blank.



Testing SOAP with SoapUI (2 of 4)



Testing SOAP with SoapUI (3 of 4)



Step 5:
Click the small green triangle
– SoapUI will send the
request over HTTP to the
IWS server!

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <xsd:getcust>
      <!--Optional:-->
      <xsd:param0>
        <!--Optional:-->
        <xsd:_CUSTNO?</xsd:_CUSTNO>
      </xsd:param0>
    </xsd:getcust>
  </soapenv:Body>
</soapenv:Envelope>
```

Testing SOAP with SoapUI (4 of 4)



Step 6:
View the returned SOAP
message (output parms) it
worked!

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <ns:getcustResponse xmlns:ns="http://getcust.com/soap/1.0">
      <ns:return>
        <ns:_CITY>FORT DODGE</ns:_CITY>
        <ns:_CUSTNO>520</ns:_CUSTNO>
        <ns:_NAME>NORTHERN LIGHTS DIST INC</ns:_NAME>
        <ns:_POSTAL>50501-5513</ns:_POSTAL>
        <ns:_STATE>IA</ns:_STATE>
        <ns:_STREET>2949 8TH AVENUE SOUTH</ns:_STREET>
      </ns:return>
    </ns:getcustResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

response time: 333ms (482 bytes)

A SOAP Service With a List



The GETCUST service only returns one "record" so to speak.
Can I do something like the "Invoice List" (the POX example) using SOAP?

- **Q:** How do I do that if I don't code the XML in the program?
- **A:** With an array!
- **Q:** How do I make an array that returns a list of "records" (more than one field per array element)?
- **A:** Use an array of data structures.
- **Q:** What if the number of returned elements (i.e. the number of invoices in the list) varies? How can I specify the number of returned array elements?
- **A:** If you code a "10i 0" parameter in your parameter list, IWS will let you use it to control the array size.

77

SOAPINV (invoice list) (1 of 2)



```
H OPTION(*SRCSTMT: *NODEBUGIO) PGMINFO(*PCML:*MODULE)
```

```
D row          ds          qualified inz
D inv          5a
D date         8s 0
D name         25a
D amount       9p 2
D weight       9p 1
```

```
D SOAPINV      PR          ExtPgm('SOAPINV')
D CustNo       4p 0 const
D strDate      8p 0 const
D endDate      8p 0 const
D rtnCount     10i 0
D rtnList      likeds(row) dim(999)
```

```
D SOAPINV      PI
D CustNo       4p 0 const
D strDate      8p 0 const
D endDate      8p 0 const
D rtnCount     10i 0
D rtnList      likeds(row) dim(999)
```

This is what needs to be returned for each invoice in the list

rtnCount will tell IWS how many invoices are returned. (to a 999 maximum)

rtnList is the returned array. Notice: LIKEDS!

78

SOAPINV (invoice list) (2 of 2)



```
rtnCount = 0;

exec SQL declare C1 cursor for
  select aiOrdn, aiIDat, aiSNme, aiDamt, aiLbs
  from ARSHIST
  where aiCust = :CustNo
  and aiIDat between :strDate
  and :endDate;

exec SQL open C1;
exec SQL fetch next from C1 into :row;

dow sqlstt='00000' or %subst(sqlstt:1:2)='01';
  rtnCount = rtnCount + 1;
  rtnList(rtnCount) = row;
  exec SQL fetch next from C1 into :row;
enddo;

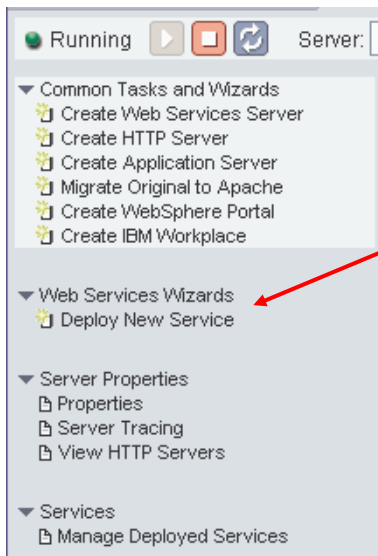
exec SQL close C1;
```

CustNo, strDate and endDate are all input parameters passed by IWS.

For each record found, rtnCount is updated, and rtnList() array contains a row.

79

SOAPINV In the Wizard (1 of 2)



Deploy new service adds another web service to the existing server.

The other screens will be the same as they were for GETCUST.

Except, that on the parameter screen, I have to tell IWS about the returned parameter count.

80

SOAPINV In the Wizard (2 of 2)



Export procedures: ?

Select	Procedure name/Parameter name	Usage	Data type	Count
<input checked="" type="checkbox"/>	▼ SOAPINV			
<input type="checkbox"/>	☒ CUSTNO	input	packed	
<input type="checkbox"/>	☒ STRDATE	input	packed	
<input type="checkbox"/>	☒ ENDDATE	input	packed	
<input type="checkbox"/>	☒ RTNCOUNT	output	int	
<input type="checkbox"/>	☒ RTNLIST	output	struct	RTNCOUNT

Select All Deselect All Expand All Collapse All

RTNCOUNT
999
RTNCOUNT

By default, the count for RTNLIST is 999, just like the DIM(999) in my RPG code.

But I can change it to "RTNCOUNT" because RTNCOUNT happens to be a 10i 0 field, IWS knows it can be used to specify the array size.

Unfortunately, there's no way to stop IWS from sending RTNCOUNT to the consumer, as well. (But if the consumer doesn't need it, it can ignore it.)

Exercises



- Download and try the REST services from this talk (using a browser.)
- Modify the REST service to take multiple parameters, and return an array (like the POX example)
- Try writing your own REST web service.
- Try the POX example. Is it easier or harder than the REST one?
- Install/configure a SOAP web service using IWS.
- Note how the IWS simplifies writing the service... but also note the performance difference, and limitations of parameter types.
- Try calling your IWS web service with SoapUI.
- **Bigger Challenge:** Using the same XML from SoapUI, and the WSDL from IWS, try creating a SOAP service manually -- like the POX one, where the data is uploaded/downloaded through Apache. Note the performance difference.

RPG as a Web Service Consumer



Presented by

Scott Klement

<http://www.scottklement.com>

© 2010-2011, Scott Klement

*"I would love to change the world, but they won't
give me the source code"*

HTTPAPI



Normally when we use the Web, we use a Web browser. The browser connects to a web server, issues our request, downloads the result and displays it on the screen.

When making a program-to-program call, however, a browser isn't the right tool. Instead, you need a tool that knows how to send and receive data from a Web server that can be integrated right into your RPG programs.

That's what HTTPAPI is for!

- HTTPAPI is a free (open source) tool to act like an HTTP client (the role usually played by the browser.)
- HTTPAPI was originally written by me (Scott Klement) to assist with a project that I had back in 2001.
- Since I thought it might be useful to others, I made it free and available to everyone.

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

More about HTTPAPI



How did HTTPAPI come about?

- I needed a way to automate downloading ACS updates from the United States Postal Service
- A friend needed a way to track packages with UPS from his RPG software
- Since many people seemed to need this type of application, I decided to make it publicly available under an Open Source license

85

Consume REST (1 of 3)



This is the REST example from the last section -- but now I'll consume it!

```
H DFTACTGRP(*NO) ACTGRP('KLEMENT') BNDDIR('HTTPAPI')
```

```
  /copy HTTPAPI_H
```

```
  /copy IFSIO_H
```

```
D url          s          1000a  varying
```

```
D stmf         s          1000a  varying
```

```
D rc           s          10i 0
```

```
D errMsg      s          52a   varying
```

```
D custInfo    ds          qualified
```

```
D id          4s 0
```

```
D name        25a
```

```
D street      25a
```

```
D city        15a
```

```
D state       2a
```

```
D postal      10a
```

```
C *ENTRY      PLIST
```

```
C             PARM          InputCust      15 5
```

86

Consume REST (2 of 3)



```
/free
  stmf = '/tmp/getcust.xml';
  url = 'http://as400.klements.com/cust/'
        + %char(%int(InputCust));

  rc = http_get(url: stmf);
  if (rc<>1 and rc<>500);
    http_crash();
  endif;

  if rc=500;
    xml-into errMsg %xml(stmf: 'path=error doc=file');
    dsply errMsg;
  else;
    xml-into custInfo %xml(stmf: 'path=result/cust doc=file');
    dsply custInfo.name;
    dsply custInfo.street;
    dsply ( custInfo.city + ' '
            + custInfo.state + ' '
            + custInfo.postal );
  endif;

  unlink(stmf);
  *inlr = *on;
/end-free
```

Consume REST (3 of 3)



When I run it like this:

```
CALL MYCUST PARM(495)
```

It responds with:

```
DSPLY ANCO FOODS
DSPLY 1100 N.W. 33RD STREET
DSPLY POMPANO BEACH FL 33064-2121
```

When I run it like this:

```
CALL MYCUST PARM(123)
```

It responds with:

```
DSPLY Unknown Customer Number
```

Currency Exchange Example



In the first section of this seminar, we talked about currency exchange, and I showed you what the SOAP messages for WebServiceX.net's currency exchange looked like.

Now it's time to try calling that web service from an RPG program!

Steps to writing a SOAP web service consumer with HTTPAPI:

- Get the WSDL
- Try the WSDL with SoapUI so you know what it looks like.
- Copy/paste the XML for the SOAP message into an RPG program.
 - ▶ Convert to a big EVAL statement
 - ▶ Insert any variable data at the right places
 - ▶ Create one big string variable with XML data.
- Pass the SOAP message to HTTPAPI's `http_post_xml()` routine.
- Parse the XML you receive as a response.

89

SOAP Consumer (1/4)



```
H DFTACTGRP(*NO) BNDDIR('LIBHTTP/HTTPAPI')

D EXCHRATE          PR                               ExtPgm('EXCHRATE')
D  Country1         3A  const
D  Country2         3A  const
D  Amount           15P 5  const
D EXCHRATE          PI
D  Country1         3A  const
D  Country2         3A  const
D  Amount           15P 5  const

/copy libhttp/qrpglesrc,httpapi_h

D Incoming          PR
D  rate             8F
D  depth            10I 0  value
D  name             1024A  varying const
D  path             24576A  varying const
D  value            32767A  varying const
D  attrs            *      dim(32767)
D                  const options(*varsize)

D SOAP              s      32767A  varying
D rc                s      10I 0
D rate              s      8F
D Result            s      12P 2
D msg               s      50A
D wait              s      1A
```

A program that uses a Web Service is called a "Web Service Consumer".

The act of calling a Web service is referred to as "consuming a web service."

SOAP Consumer (2/4)



Constructing the SOAP message is done with a big EVAL statement.

```
/free
SOAP =
'<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>'
+ '<SOAP:Envelope'
+ '   xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"'
+ '   xmlns:tns="http://www.webserviceX.NET/">'
+ '<SOAP:Body>'
+ '   <tns:ConversionRate>'
+ '     <tns:FromCurrency>'+ %trim(Country1) + '</tns:FromCurrency>'
+ '     <tns:ToCurrency>'+ %trim(Country2) + '</tns:ToCurrency>'
+ '   </tns:ConversionRate>'
+ '</SOAP:Body>'
+ '</SOAP:Envelope>';

rc = http_post_xml(
'http://www.webserviceX.net/CurrencyConvertor.asmx'
: %addr(SOAP) + 2
: %len(SOAP)
: *NULL
: %paddr(Incoming)
: %addr(rate)
: HTTP_TIMEOUT
: HTTP_USERAGENT
: 'text/xml'
: 'http://www.webserviceX.NET/ConversionRate');
```

This routine tells HTTPAPI to send the SOAP message to a Web server, and to parse the XML response.

As HTTPAPI receives the XML document, it'll call the INCOMING subprocedure for every XML element, passing the "rate" variable as a parameter.

91

SOAP Consumer (3/4)



If an error occurs, ask HTTPAPI what the error is.

```
if (rc <> 1);
  msg = http_error();
else;
  Result = %dech(Amount * rate: 12: 2);
  msg = 'Result = ' + %char(Result);
endif;

dsply msg ' ' wait;

*inlr = *on;

/end-free

P Incoming      B
D Incoming      PI
D rate          8F
D depth        10I 0 value
D name         1024A varying const
D path        24576A varying const
D value       32767A varying const
D attrs       * dim(32767)
D              const options(*varsize)

/free
  if (name = 'ConversionRateResult');
    rate = %float(value);
  endif;
/end-free

P              E
```

Display the error or result on the screen.

This is called for every XML element in the response.

When the element is a "Conversion Rate Result" element, save the value, since it's the exchange rate we're looking for!

SOAP Consumer (4/4)



Here's a sample of the output from calling the preceding program:

```
Command Entry                                     Request level: 1

Previous commands and messages:
> call exchrates parm('USD' 'EUR' 185.50)
DSPLY Result = 133.69

Bottom
Type command, press Enter.
===>

F3=Exit    F4=Prompt    F9=Retrieve    F10=Include detailed messages
F11=Display full    F12=Cancel    F13=Information Assistant    F24=More keys
```

93

What Just Happened?



HTTPAPI does not know how to create an XML document, but it does know how to parse one.

In the previous example:

- The SOAP document was created in a variable using a big EVAL statement.
- The variable that contained the SOAP document was passed to HTTPAPI and HTTPAPI sent it to the Web site.
- The subprocedure we called (`http_post_xml`) utilizes HTTPAPI's built-in XML parser to parse the result as it comes over the wire.
- As each XML element is received, the `Incoming()` subprocedure is called.
- When that subprocedure finds a `<ConversionRateResult>` element, it saves the element's value to the "rate" variable.
- When `http_post_xml()` has completed, the rate variable is set. You can multiply the input currency amount by the rate to get the output currency amount.

94

No! Let Me Parse It!



If you don't want to use HTTPAPI's XML parser, you can call the `http_url_post()` API instead of `http_post_xml()`.

In that situation, the result will be saved to a stream file in the IFS, and you can use another XML parser instead of the one in HTTPAPI.

```
....
rc = http_url_post(
    'http://www.webserviceX.net/CurrencyConvertor.asmx'
    : %addr(SOAP) + 2
    : %len(SOAP)
    : '/tmp/CurrencyExchangeResult.soap'
    : HTTP_TIMEOUT
    : HTTP_USERAGENT
    : 'text/xml'
    : 'http://www.webserviceX.NET/ConversionRate' );
....
```

For example, you may want to use RPG's built in support for XML in V5R4 to parse the document rather than let HTTPAPI do it. (XML-SAX op-code)

95

Handling Errors with HTTPAPI



Most of the HTTPAPI routines return 1 when successful

- Although this allows you to detect when something has failed, it only tells you *that* something failed, not *what* failed
- The `http_error()` routine can tell you an error number, a message, or both
- The following is the prototype for the `http_error()` API

```
D http_error      PR          80A
D   peErrorNo    10I 0 options(*nopass)
```

The human-readable message is particularly useful for letting the user know what's going on.

```
if ( rc <> 1 );
    msg = http_error();
    // you can now print this message on the screen,
    // or pass it back to a calling program,
    // or whatever you like.
endif;
```

96

Handling Errors, continued...



The error number is useful when the program anticipates and tries to handle certain errors.

```
if ( rc <> 1 );

    http_error(errnum);

    select;
    when errnum = HTTP_NOTREG;
        // app needs to be registered with
        exsr RegisterApp;
    when errnum = HTTP_NDAUTH;
        // site requires a userid/password
        exsr RequestAuth;
    other;
        msg = http_error();
    endsl;

endif;
```

These are constants that are defined in HTTPAPI_H (and included with HTTPAPI)

97

WSDL2RPG



Instead of SoapUI, you might consider using WSDL2RPG – another open source project, this one from Thomas Raddatz. You give WSDL2RPG the URL or IFS path of a WSDL file, and it generates the RPG code to call HTTPAPI.

```
WSDL2RPG URL('/home/klemscot/CurrencyConvertor.wsdl')
          SRCFILE(LIBSCK/QRPGLESRC)
          SRCMBR(CURRCONV)
```

Then compile CURRCONV as a module, and call it with the appropriate parameters.

- Code is still beta, needs more work.
- The RPG it generates often needs to be tweaked before it'll compile.
- The code it generates is much more complex than what you'd use if you generated it yourself, or used SoapUI
- Can only do SOAP (not POX or REST)

But don't be afraid to help with the project! It'll be really nice when it's perfected!

<http://www.tools400.de/English/Freeware/WSDL2RPG/wsdl2rpg.html>

98

About SSL with HTTPAPI



The next example (UPS package tracking) requires that you connect using SSL. (This is even more important when working with a bank!)

HTTPAPI supports SSL when you specify "https:" instead of "http:" at the beginning of the URL.

It uses the SSL routines in the operating system, therefore you must have all of the required software installed. IBM requires the following:

- Digital Certificate Manager (option 34 of OS/400, 57xx-SS1)
- TCP/IP Connectivity Utilities for iSeries (57xx-TC1)
- IBM HTTP Server for iSeries (57xx-DG1)
- IBM Developer Kit for Java (57xx-JV1)
- IBM Cryptographic Access Provider (5722-AC3) *(pre-V5R4 only)*

Because of (historical) import/export laws, 5722-AC3 is not shipped with OS/400. However, it's a no-charge item. You just have to order it separately from your business partner. It is included automatically in V5R4 and later as 57xx-NAE

99

UPS Example (slide 1 of 11)



This demonstrates the "UPS Tracking Tool" that's part of UPS OnLine Tools. There are a few differences between this and the previous example:

- You have to register with UPS to use their services (but it's free)
- You'll be given an access key, and you'll need to send it with each request.
- UPS requires SSL to access their web site.
- UPS does not use SOAP or WSDL for their Web services – but does use XML. Some folks call this "Plain Old XML" (POX).
- Instead of WSDL, they provide you with documentation that explains the format of the XML messages.
- That document will be available from their web site after you've signed up as a developer.

100

UPS Example (slide 2 of 11)



```
tn5250 - W7 - ssl:as400.klements.com
File Edit Help
Track UPS Package

Enter Tracking No: 1ZE15A564232247639

F3=Exit
5250
```

052/012

101

UPS Example (slide 3 of 11)



```
tn5250 - W7 - ssl:as400.klements.com
File Edit Help
11/16/04 Track UPS Package 1ZE15A564232247639
Signed By DENNIS

Status      Date      Time      City      St      Description
DELIVERED   11/09/2004 11:54:00 MILWAUKEE WI DOCK
OUT FOR DELIVERY 11/09/2004 07:10:00 OAK CREEK WI
ARRIVAL SCAN 11/08/2004 23:59:00 OAK CREEK WI
DEPARTURE SCAN 11/08/2004 21:55:00 HODGKINS IL
ORIGIN SCAN 11/08/2004 12:00:05 HODGKINS IL
BILLING INFORMATION 11/07/2004 10:56:54

5250
```

001/001

102

UPS Example (slide 4 of 11)



```
...
D UPS_USERID      C      '<put your userid here>'
D UPS_PASSWD     C      '<put your password here>'
D UPS_LICENSE     C      '<put your access license here>'
...

d act            s      10I 0
d activity       ds      qualified
d               dim(10)
d Date           8A
d Time           6A
D Desc           20A
D City           20A
D State          2A
D Status         20A
D SignedBy      20A
...

// Ask user for tracking number.
exfmt TrackNo;
```

UPS provides these when you sign up as a developer.

UPS Example (slide 5 of 11)



```
postData =
'<?xml version="1.0"?>'
'<AccessRequest xml:lang="en-US">'
'  <AccessLicenseNumber>' + UPS_LICENSE + '</AccessLicenseNumber>' +
'  <UserId>' + UPS_USERID + '</UserId>'
'  <Password>' + UPS_PASSWD + '</Password>'
'</AccessRequest>'
'<?xml version="1.0"?>'
'<TrackRequest xml:lang="en-US">'
'  <Request>'
'    <TransactionReference>'
'      <CustomerContext>Example 1</CustomerContext>'
'      <XpciVersion>1.0001</XpciVersion>'
'    </TransactionReference>'
'    <RequestAction>Track</RequestAction>'
'    <RequestOption>activity</RequestOption>'
'  </Request>'
'  <TrackingNumber>' + TrackingNo + '</TrackingNumber>'
'</TrackRequest>'

rc = http_post_xml('https://wwwcie.ups.com/ups.app/xml/Track'
                  : %addr(postData) + 2
                  : %len(postData)
                  : %paddr(StartOfElement)
                  : %paddr(EndOfElement)
                  : *NULL );

if (rc <> 1);
  msg = http_error();
  // REPORT ERROR TO USER
endif;
```

The StartOfElement and EndOfElement routines are called while http_post_xml is running

UPS Example (slide 6 of 11)



```
for RRN = 1 to act;
monitor;
tempDate = %date(activity(RRN).date: *ISO0);
scDate = %char(tempDate: *USA);
on-error;
scDate = *blanks;
endmon;

monitor;
tempTime = %time(activity(RRN).time: *HMS0);
scTime = %char(tempTime: *HMS);
on-error;
scTime = *blanks;
endmon;

scDesc = activity(RRN).desc;
scCity = activity(RRN).city;
scState = activity(RRN).state;
scStatus = activity(RRN).status;

if (scSignedBy = *blanks);
scSignedBy = activity(RRN).SignedBy;
endif;

write SFLREC;
endfor;
```

Since the StartOfElement and EndOfElement routines read the XML data and put it in the array, when http_post_xml is complete, we're ready to load the array into the subfile.

105

UPS Example (slide 7 of 11)



```
<?xml version="1.0" ?>
<TrackResponse>
  <Shipment>
    . . .
    <Package>
      <Activity>
        <ActivityLocation>
          <Address>
            <City>MILWAUKEE</City>
            <StateProvinceCode>WI</StateProvinceCode>
            <PostalCode>53207</PostalCode>
            <CountryCode>US</CountryCode>
          </Address>
          <Code>AI</Code>
          <Description>DOCK</Description>
          <SignedForByName>DENNIS</SignedForByName>
        </ActivityLocation>
        <Status>
          <StatusType>
            <Code>D</Code>
            <Description>DELIVERED</Description>
          </StatusType>
          <StatusCode>
            <Code>KB</Code>
          </StatusCode>
        </Status>
        <Date>20041109</Date>
        <Time>115400</Time>
      </Activity>
```

This is what the response from UPS will look like.

HTTPAPI will call the StartOfElement procedure for every "start" XML element.

HTTPAPI will call the EndOfElement procedure for every "end" XML element. At that time, it'll also pass the value.

106

UPS Example (slide 8 of 11)



```
<Activity>
  <ActivityLocation>
    <Address>
      <City>OAK CREEK</City>
      <StateProvinceCode>WI</StateProvinceCode>
      <CountryCode>US</CountryCode>
    </Address>
  </ActivityLocation>
  <Status>
    <StatusType>
      <Code>I</Code>
      <Description>OUT FOR DELIVERY</Description>
    </StatusType>
    <StatusCode>
      <Code>DS</Code>
    </StatusCode>
  </Status>
  <Date>20041109</Date>
  <Time>071000</Time>
</Activity>
. . .
</Package>
</Shipment>
</TrackResponse>
```

There are additional <Activity> sections and other XML that I omitted because it was too long for the presentation.

107

UPS Example (slide 9 of 11)



```
P StartOfElement B
D StartOfElement PI
D UserData * value
D depth 10I 0 value
D name 1024A varying const
D path 24576A varying const
D attrs * dim(32767)
D const options(*varsize)
/free
if path = '/TrackResponse/Shipment/Package' and name='Activity';
act = act + 1;
endif;
/end-free
P E
```

This is called during `http_post_xml()` for each start element that UPS sends. It's used to advance to the next array entry when a new package record is received.

108

UPS Example (slide 10 of 11)



```
P EndOfElement      B
D EndOfElement      PI
D UserData           *   value
D depth             10I 0 value
D name              1024A varying const
D path              24576A varying const
D value             32767A varying const
D attrs            *   dim(32767)
D                   const options(*varsize)
/free

select;
when path = '/TrackResponse/Shipment/Package/Activity';

    select;
    when name = 'Date';
        activity(act).Date = value;
    when name = 'Time';
        activity(act).Time = value;
    ends;

when path = '/TrackResponse/Shipment/Package/Activity' +
            '/ActivityLocation';

    select;
    when name = 'Description';
        activity(act).Desc = value;
    when name = 'SignedForByName';
        activity(act).SignedBy = value;
    ends;
ends;
```

This is called for each ending value. We use it to save the returned package information into an array.

Remember, this is called by http_post_xml, so it'll run before the code that loads this array into the subfile!

109

UPS Example (slide 11 of 11)



```
when path = '/TrackResponse/Shipment/Package/Activity' +
            '/ActivityLocation/Address';

select;
when name = 'City';
    activity(act).City = value;
when name = 'StateProvinceCode';
    activity(act).State = value;
ends;

when path = '/TrackResponse/Shipment/Package/Activity' +
            '/Status/StatusType';

if name = 'Description';
    activity(act).Status = value;
endif;

ends;

/end-free
P                                     E
```

110

Exercises



- Find a REST web service on the Internet, and try consuming it with HTTPAPI.
- Such as xurrency.
- I demonstrated POX by tracking a package with UPS. Try writing one that consumes the POX RPG example from the "providing" section.
- I demonstrated SOAP with the WebServiceX.net currency exchange service. Try writing a SOAP consumer that calls the GETCUST example from the "providing" section.

111

HTTPAPI Information



You can download *HTTPAPI* from Scott's Web site:

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

Most of the documentation for *HTTPAPI* is in the source code itself.

- Read the comments in the HTTPAPI_H member
- Sample programs called EXAMPLE1 - EXAMPLE20

The best places to get help for *HTTPAPI* are:

- the FTPAPI/HTTPAPI mailing list
 - Signup: <http://www.scottklement.com/mailman/listinfo/ftpapi>
 - Archives: <http://www.scottklement.com/archives/ftpapi/>
- the System iNetwork Forums
 - <http://www.systeminetwork.com/forums>

112

More Information / Resources



Gaining a basic understanding of HTTP:

What Is HTTP, Really? (Scott Klement)

<http://systeminetwork.com/article/what-http-really>

What's the Difference Between a URI, URL, and Domain Name? (Scott Klement)

<http://www.systeminetwork.com/article/application-development/whats-the-difference-between-a-uri-url-and-domain-name-65224>

Gaining a basic understanding of Web Services & Terminology:

Web Services: The Next Big Thing (Scott N. Gerard)

<http://www.systeminetwork.com/article/other-languages/web-services-the-next-big-thing-13626>

SOAP, WDSL, HTTP, XSD? What? (Aaron Bartell)

<http://systeminetwork.com/article/soap-wdsl-http-xsd-what>

113

More Information / Resources



w3schools.com -- free (and great!) site for learning web technology

XML: <http://www.w3schools.com/xml/default.asp>

Web Services: <http://www.w3schools.com/webservices/default.asp>

WSDL: <http://www.w3schools.com/wSDL/default.asp>

SOAP: <http://www.w3schools.com/soap/default.asp>

IBM's web site for the Integrated Web Services (IWS) tool:

<http://www.ibm.com/systems/i/software/iws/>

http://www.ibm.com/systems/i/software/iws/quickstart_server.html

SoapUI home page

<http://www.soapui.org>

WSDL2RPG Home Page

<http://www.tools400.de/English/Freeware/WSDL2RPG/wsd2rpg.html>

Call a Web Service with WSDL2RPG (Thomas Raddatz)

<http://systeminetwork.com/article/call-web-service-wsd2rpg>

114

More Information / Resources



How-To Articles About Consuming/Providing Web Services:

RPG Consumes the REST (Scott Klement)

<http://systeminetwork.com/article/rpg-consumes-rest>

RPG Consuming Web Services with HTTPAPI and SoapUI (Scott Klement)

<http://systeminetwork.com/article/rpg-consuming-web-services-httpapi-and-soapui>

IBM's Integrated Web Services (Scott Klement)

<http://systeminetwork.com/article/ibms-integrated-web-services>

Consume Web Services with IBM's IWS (Scott Klement)

<http://www.systeminetwork.com/article/rpg-programming/consume-web-services-with-ibms-iws-66209>

UPS OnLine Tools

http://www.ups.com/e_comm_access/gettools_index

115

More Information / Resources



Sites that offer web service directories

- WebServiceX.net
- XMethods.net
- BindingPoint.com
- RemoteMethods.com

RPG's XML Opcodes & BIFs:

"Real World" Example of XML-INTO (Scott Klement)

<http://systeminetwork.com/article/real-world-example-xml>

RPG's XML-SAX Opcode

<http://systeminetwork.com/article/rpgs-xml-sax-opcode>

PTFs for Version 6.1 Enhance RPG's XML-INTO

<http://systeminetwork.com/article/ptfs-version-61-enhance-rpgs-xml>

XML-INTO: Maximum Length

<http://systeminetwork.com/article/xml-maximum-length>

XML-INTO: Read XML Data Larger Than 65535

<http://systeminetwork.com/article/xml-read-xml-data-larger-65535>

XML-INTO: Output to Array Larger than 16 MB

<http://systeminetwork.com/article/xml-output-array-larger-16-mb>

116

This Presentation



You can download a PDF copy of this presentation from:

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

*The Sample Web Service Providers/Consumers in this article
are also available at the preceding link.*

Thank you!