Web Services and XML for RPGers

COMMON PID 570176
(Saturday Workshop, Monorail C)

Presented by

Scott Klement

http://www.scottklement.com

"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 six parts:

1. XML Terminology and Basics
2. How to Write XML from RPG
3. How to Read XML with RPG's XML Opcodes
4. Web Services Terminology and Basics
5. How to Provide RPG-based Web Services
6. How to Call Web Services from RPG
   (with HTTPAPI)

What is XML?

XML stands for

- eXtensible Markup Language
- A "markup language" is for labelling data
  - like "markings" on the outside of a box to tell what's inside
  - in this case, however, the markings are *inside* the data.

It's extensible because you can make up your own tags, as needed.

You can add additional tags later, without breaking compatibility.

XML Does Not Do Anything

XML does not *do* anything.

- Although "language" is part of the name, it is not a programming language like RPG.
- It is only a format for storing/retrieving data
- It looks similar to HTML
- It is designed to carry data, not display it.
- XML tags are not "predefined" like HTML
- You invent your own XML tags.
  - or use the ones prescribed by someone else.
  - A language for creating other languages

Usually Used for Data Interchange

EDI? Data synchronization? Mailing list exchanges? Employee information for a benefit plan?

Any data that you need to exchange between organizations.

Imagine you had to send a list of orders to a business partner. Why not use a physical file for this?

- Multiple files required
  - one containing one record per order (for "header-level" data)
  - one containing one record per item/service on the order
  - possibly more (multiple ship-tos? multiple discounts? etc.)

- All computer systems and databases use different formats
- Data description (field layouts) must be sent separately.
- Instructions for how the files correlate must be sent separately.
- If you add more information later, all business partners must change their applications or compatibility is broken.

The XML Solution

- You present your data with "tags" that identify what it is.
  - No need for a separate description

- Any sort of data can be represented
  - No worries about multiple documents to contain all of the data

- You can add new tags without breaking compatibility.
  - Exising applications will just ignore new data until updated.

- All modern programming languages have XML reading capability
  - Java, .NET, PHP, JavaScript, RPG, Cobol, C/C++, etc.
  - … sometimes an add-on software package is required.

To better understand how this is accomplished, let's look at the syntax of an XML document.

XML Elements and Attributes

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>

Simple XML Example

```
<Customer id="56071" type="retail">
   <name>Bountiful Grocery, Inc.</name>
   <contact>Joseph Bachmann</contact>
   <address>
     <street>535 N Wabash Ave</street>
     <city>Chicago</city>
     <state>IL</state>
     <postal>60611</postal>
   </address>
</Customer>
```

- Customer is an *element* with an opening and closing *tag*
- The `<Customer>` opening tag has *attributes* of `id` and `type`
- <name> is within the start/end of `<Customer>`.
- `<street>` is within address, which in turn, is within `<Customer>`

Elements and Tags

- All XML elements must have a closing tag
  - In HTML or SGML, some tags do not have to be closed.
  - In XML, they always do.

    `<Customer>` must have a corresponding `</Customer>`

    But: `<Customer />` is the same as `<Customer></Customer>`
- XML tags are CaSe SenSitiVe
  - Valid: `<street>123 Main St.</street>`
  - Invalid: `<Street>123 Main St.</street>`
- XML elements may contain nested XML elements (but must be properly nested)
  - Valid:

    `<address><street>123 Main St.</street></address>`
  - Invalid:

    `<address><street>123 Main St.</address></street>`

Attributes

```
<address usage="BillTo">
```

- Attributes are intended to:
  - *Clarify* or *qualify* the meaning of a tag
  - …but the distinction is often unclear/blurry.
  - …and there are no official rules about when to use an attribute vs. a nested element.
- Attributes must be quoted. (Either single or double quotes can be used.)
- Attributes *cannot* contain any nested elements
- Attributes *cannot* contain multiple values or lists.
- When double quotes are used, single quotes can be data. Or vice-versa.

```
<note subject="You're Crazy!">
```

```
<actor name='George "Spanky" McFarland'>
```

Which Do You Like Better?

```
<message date="04-10-2011">
  Hello there! Nice day, isn't it?
</message>
```

```
<message>
  <date>04-10-2011</date>
  Hello there! Nice day, isn't it?
</message>
```

```
<message>
  <date>
    <month>04</month>
    <day>10</day>
    <year>2011</year>
  </date>
  Hello there! Nice day, isn't it?
</message>
```

XML Documents

- Must have a root element (element that encloses the entire document)

VALID:

```
<document name="customers">
  <customer id="1001" />
  <customer id="1002" />
</document>
```

NOT VALID:

```
<customer id="1001" />
<customer id="1002" />
```

- Are not organized into "records".
- Line feeds are optional

```
<document name="customers"><customer id="1001" /></document>
```

- May be megabytes or gigabytes long. (without linefeeds!)
- Are usually encoded in Unicode. Or sometimes ASCII. Rarely EBCDIC.

A More Real-World Example

```
<orderList>
  <order number="12345">
    <date type="order">2011-04-10</date>
    <date type="ship">2011-05-07</date>
    <Customer id="56071" type="retail">
      <name>Bountiful Grocery, Inc.</name>
      <contact>Joseph Bachmann</contact>
      <address type="shipto"><street>535 N Wabash Ave</street>
        <city>Chicago</city><state>IL</state><postal>60611</postal>
      </address>
    </Customer>
    <itemList>
      <item gtin="007360804430">
        <name>Italian Sausage</name>
        <ordered qty="150" unit="case"/>
      </item>
      <item gtin="007360801790">
        <name>Olive Loaf</name>
        <ordered qty="75" unit="loaf"/>
      </item>
    </itemList>
  </order>
  <order>More Orders Can be Here!</order>
</orderList>
```

It's okay for the same element name to be used more than once.

More than one element on a line is allowed and very common place.

Elements may be repeated to form a "list" (or "array")

Processing Directives

```
<?xml version="1.0" encoding="utf-8"?>
<orderList>
  <order number="12345">
     ... etc ...
  </order>
</orderList>
```

- have question marks like <? and ?> to disguise them from normal tags
- never have a closing tag
- provide hints to XML parsers to help them understand the document
  - this document uses version 1.0 of the XML specification
  - this document is not in EBCDIC or ASCII, but rather UTF-8 Unicode
- are not part of the data of your program
- XML parsers typically do not return processing information to the programs that call them. (Or when they do, it's provided separately.)

Special Characters

```
<note>If Calories < 100 Then</note>
```

This will confuse an XML parser. It will expect a new tag to start here.

- Almost any character may be used in an XML document
- This includes international characters (especially if a Unicode encoding was chosen.)
- The only characters that are strictly forbidden are:
  - < because it starts a new tag
  - & because it starts a new entity reference
- But it's a good idea to also escape these
  - " especially when used inside an attribute
  - ' especially when used inside an attribute
  - > because it ends a tag.

Entity References

An *Entity Reference* lets you 'escape' special characters in XML. There are 5 entities that are predefined in every XML parser:

- `&lt;` refers to the < character
- `&gt;` refers to the > character
- `&amp;` refers to the & character
- `&quot;` refers to the " character
- `&apos;` refers to the ' character

NOTE: It's also possible to define your own entities -- but this is rarely done in XML, so I will not cover it here.

```
<note>If Calories &lt; 100 Then</note>

<actor name="George &quot;Spanky&quot; McFarland">
```

Comments in XML

It's possible to put comments in an XML document (much like you would in RPG source code.)

```
<my_document>
  <!-- This is a comment. -->
  <data> some data here </data>
</my_document>
```

Comments always start with `<!--`

Comments always end with `-->`

CDATA (and PCDATA)

The data in-between XML tags is considered the "character data"

```
<my_tag> ...Character Data ... </my_tag>
```

There are two types of character data allowed in XML:

- **PCDATA = Parsed Character Data (default), the data is parsed looking for other XML tags, attributes and entities.**
- **CDATA = Character data. The data does not contain any XML tags, attributes or entities.**

CDATA is particularly useful when your data contains a lot of special symbols (such as < or & characters.) This is especially true of program code.

```
<code language="clle">
<![CDATA[
   DCL VAR(&USERID) TYPE(*CHAR) LEN(10)
   RTVJOBA CURUSER(&USERID)
]]>
</code>
```

No need to escape the & characters because they're in a CDATA block

The only sequence of characters not allowed in CDATA is `]]>` since it denotes the end of the CDATA.

Well Formed

An XML document is said to be *well formed* when it follows basic XML syntax rules.

- all elements are within a root element.
- all elements have a closing tag
- all tags are case-sensitive
- all elements are properly nested
- all element values are properly quoted
- all special characters in the data are converted to entity references

That's not the same thing as being a *valid document*. When you (or someone you do business with) designs their own document layout, it may have additional rules.

- certain elements may be required?
- elements/attributes might have to be in a certain order?
- data might have to be in a certain format?

Line Breaks Optional

The XML standard does not require your document to be laid and formatted nicely. It's very common for XML documents to have no line breaks at all. All of the tags appear on a single line!

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<Customer id="56071" type="retail"><name>Bountiful Grocery,
Inc.</name><contact>Joseph Bachmann</contact><address>
<street>535 N Wabash Ave</street><city>Chicago</city>
<state>IL</state><postal>60611</postal></address></Customer>
```

This is still considered well-formed! And it may even be considered "valid"!

Example of Not Well Formed

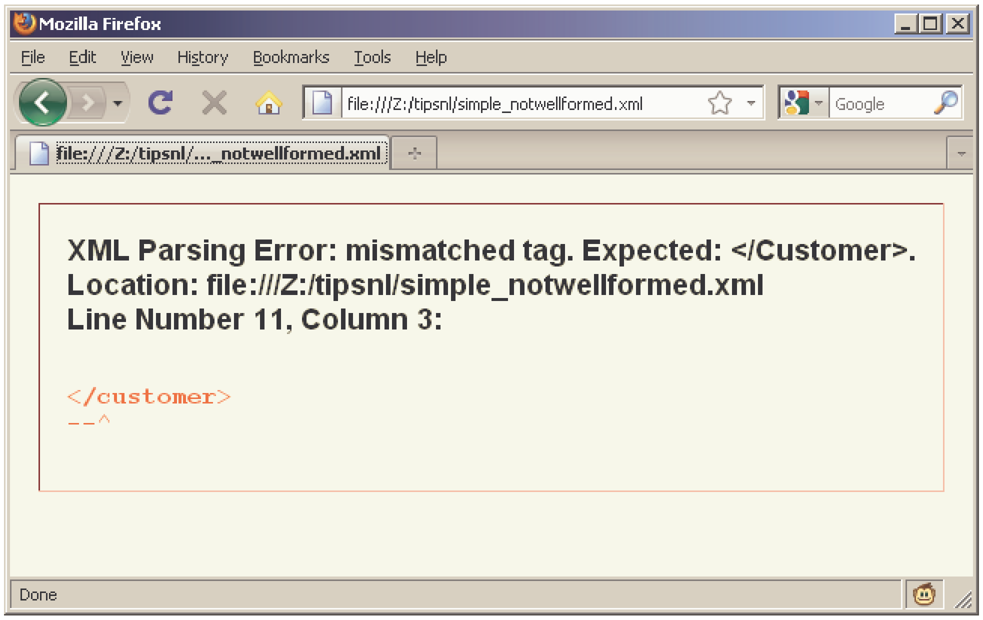
```
<Customer id="56071" type="retail">
   <name>Bountiful Grocery, Inc.</name>
   <contact>Joseph Bachmann</contact>
   <address>
     <street>535 N Wabash Ave</street>
     <city>Chicago</city>
     <state>IL</state>
     <postal>60611</postal>
   </address>
</customer>
```

Not well formed because `<Customer>` does not match `</customer>`

Q: When your RPG program outputs XML, you might make a mistake in your code. How do you test to see if your document is well-formed?

Interactive Test with Browser

A: Open your XML with a browser (Firefox, Internet Explorer, Chrome, Safari) and it'll tell you if it's well-formed or not.

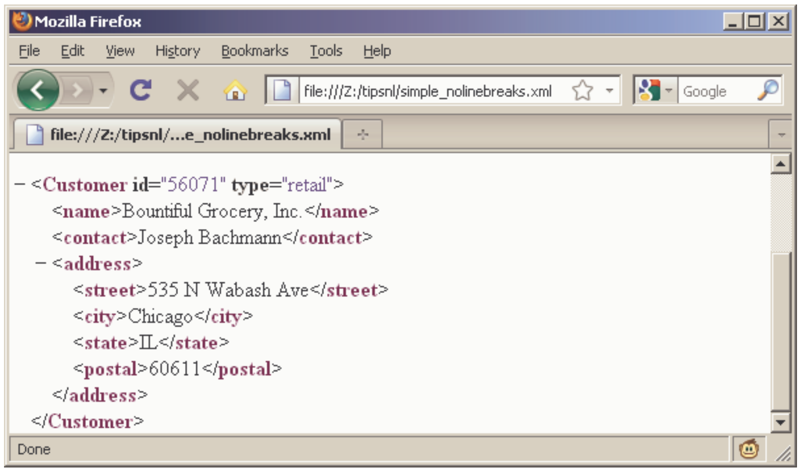

Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///Z:/tipsnl/simple_notwellformed.xml
Google
file:///Z:/tipsnl/..._notwellformed.xml
Done

```
XML Parsing Error: mismatched tag. Expected: </Customer>.
Location: file:///Z:/tipsnl/simple_notwellformed.xml
Line Number 11, Column 3:

</customer>
--^
```

Browsers Make It Readable

Side Note: When you have a document with no line breaks, a browser also helps make it easy to read.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?><Customer id="56071"
type="retail"><name>Bountiful Grocery, Inc.</name><contact>Joseph Bachmann
</contact><address><street>535 N Wabash Ave</street><city>Chicago</city><state>
IL</state><postal>60611</postal></address></Customer>
```


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///Z:/tipsnl/simple_nolinebreaks.xml
Google
file:///Z:/tipsnl/...e_nolinebreaks.xml
Done

```
- <Customer id="56071" type="retail">
    <name>Bountiful Grocery, Inc.</name>
    <contact>Joseph Bachmann</contact>
  - <address>
      <street>535 N Wabash Ave</street>
      <city>Chicago</city>
      <state>IL</state>
      <postal>60611</postal>
    </address>
  </Customer>
```

Programmatic Test With Parser

```
    monitor;
       xml-sax %handler(checkDoc : errorCode)
               %XML( 'simple_notwellformed.xml'
                   : 'doc=file' );
    on-error 351;
       // RNQ0351: The XML parser detected error code &1.
       // not well-formed! ‘errorCode’ contains the error code.
    endmon;


 P checkDoc         B
 D                  PI            10i 0
 D   errorCode                    10i 0
 D   event                        10i 0 value
 D   string                         *   value
 D   stringLen                    20i 0 value
 D   exceptionId                  10i 0 value
  /free
     if event = *XML_EXCEPTION;
        errorCode = exceptionId;
        return 1;
     endif;
     return 0;
  /end-free
 P                  E
```

If you want to check if a document is well formed within a progam, you can simply run it through the parser.

XML standards say that an XML parser must return an error if a document is not well-formed.

RPG's XML-SAX opcode sends you an exception ID via it's handler.

Name Spaces

Since you can make up any XML tags you want/need for your document, how do you avoid naming conflicts?

Suppose you wanted to mix your XML document with one created by someone else. But, you *don't want duplicated names*?

```
<Customer>
  <name>John Q Public</name>
</Customer>
```

```
<Product>
  <name>Bratwurst</name>
</Product>
```

```
<Order>
  <Customer><name>John Q Public</name></Customer>
  <productList>
     <Product><name>Bratwurst</name></Product>
     <Product><name>Kielbasa</name></Product>
  </productList>
</Order>
```

Prefixes Make it Unique

Suppose you added a prefix for each document.

- c = from the Customer document
- p = from the Product document

```
<Order>
  <c:Customer><c:name>John Q Public</c:name></c:Customer>
  <p:productList>
     <p:Product><p:name>Bratwurst</p:name></p:Product>
     <p:Product><p:name>Kielbasa</p:name></p:Product>
  </p:productList>
</Order>
```

`<c:name>` is clearly the customer name.

`<p:name>` is clearly the product name.

These prefixes are called name spaces.

Globally Unique

Most XML documents that use name spaces strive to make them globally unique.

Remember: Anyone in the world can create his/her own XML tags. How can you guarantee that something is identified uniquely?

How would you do it?

- *make people register their XML elements and name spaces?*
- *allow only licensed operators to create XML tags? Use their license number as a prefix?*

These sort of defeat the purpose of letting people create their own tags, don't they?

How would you keep them unique?

Use a URL as a Prefix

The standard allows you to use a URL as a name space.

- Anyone can register a domain name (e.g. scottklement.com)
- But only one person or organization can own a particular name
- Most organizations already own one.
- The remainder of the URL (besides the domain name) can allow multiple name spaces to be offered by a single company.

```
<Order xmlns:c="http://systeminetwork.com/ns/customer"
       xmlns:p="http://scottklement.com/xml/productNs" >

  <c:Customer><c:name>John Q Public</c:name></c:Customer>
  <p:productList>
     <p:Product><p:name>Bratwurst</p:name></p:Product>
     <p:Product><p:name>Kielbasa</p:name></p:Product>
  </p:productList>

</Order>
```

The XMLNS Attribute

The `xmlns` attribute is a reserved word for XML Name Space. It lets you specify a prefix and a URL.

- The URL is only for uniqueness! Nothing is fetched over the network or Internet for the URL.
- Does *not* have to point to a real document on a real web server.
- But, many think it's a good idea to have the URL link to the document's schema (XSD) file. (I'll explain schemas soon!)

```
<Order xmlns:c="http://systeminetwork.com/ns/customer"
       xmlns:p="http://scottklement.com/xml/productNs" >

... Because xmlns is specified on the 'Order' element, the c: and p: namespaces
    are only valid between <Order> and </Order> (but can be used for the
    order tag itself) ...

</Order>
```

Default and Duplicated XMLNS

- When `xmlns` is specified without a prefix, it's the default namespace within that tag.
- It's possible (but confusing and not recommended) to have more than one way of referring to the same name space.

```
<Order xmlns="http://scottklement.com/xmldemo/xmldemo/order"
       xmlns:cust="http://systeminetwork.com/ns/customer"
       xmlns:prod="http://scottklement.com/xmldemo/order" >

   <cust:Customer> customer data here </cust:Customer>
   <date>2011-04-11</date>

   <productList>
     <prod:Product>  product data here  </prod:Product>
     <Product> More here </Product>
   </productList>
</Order>
```

`Order, date, productList and Product` (both of them) all use the same name space. `prod:Product` and `Product` should be considered the same.

But, `Customer` uses a different name space.

XML Schema Documents

XML is a *"markup language for creating new markup languages"*

When you've designed your own language, or have to conform to someone else's, how can you test that your document is not only well-formed, but also valid?

- to be *valid*, a document must also be well-formed.
- it must also conform to the rules dictated by the document creator
  - which elements are allowed?
  - which elements are manditory?
  - how many times can a loop repeat?
  - in what sequence do the elements/attributes need to appear?
    - or, doesn't it matter?

XML rules can be described with DTD (old, deprecated) or XSD documents

(Due to time constraints, I will only give a quick primer for XSD in this talk.)

XML Schema Documents

XML is a *"markup language for creating new markup languages"*

When you've designed your own language, or have to conform to someone else's, how can you test that your document is not only well-formed, but also valid?

- to be *valid*, a document must also be well-formed.
- it must also conform to the rules dictated by the document creator
  - which elements are allowed?
  - which elements are manditory?
  - how many times can a loop repeat?
  - in what sequence do the elements/attributes need to appear?
    - or, doesn't it matter?

XML rules can be described with DTD (old, deprecated) or XSD documents

(Due to time constraints, I will only give a quick primer for XSD in this talk.)

XSD Example

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Customer">
    <xs:complexType>
      <xs:all>
        <xs:element name="name"    type="xs:string"/>
        <xs:element name="contact" type="xs:string"/>
        <xs:element name="address">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="street" type="xs:string"/>
              <xs:element name="city"   type="xs:string"/>
              <xs:element name="state"  type="xs:string"/>
              <xs:element name="postal" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:all>
      <xs:attribute name="id"   type="xs:string"
          use="required"/>
      <xs:attribute name="type" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

An XSD is an XML document.

It lists which elements and attrs are allowed within an XML document.

The xs: prefix is a namespace.

<xs:all> allows the elements in any order

<xs:sequence> requires the elements to be in a particular order.

Other Advanced XML Terms

- XSL = eXtensible Stylesheet Language = tells a browser how to display your XML document. (Similar to CSS in HTML.)
  - XSLT = XSL for transformations.
  - XSL-FO = XSL formatting into objects
- XPath = path of elements needed to reach a target

  Example: `/Customer/address/city`
- SAX = Simple API for XML

  A type of XML parser that walks through your document, one element at a time.
- DOM = Document Object Model

  The opposite of SAX. Loads entire document into memory. You navigate by refering to parent, child and sibling nodes.

This Presentation

You can download a PDF copy of this presentation from:

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

Thank you!

Writing XML from RPG

Presented by

Scott Klement

http://www.scottklement.com

Apprentice: *“Master, what is fate?”*

Mentor: *“It is that which brings medicine to the sick. Food to the hungry. Materials to those who would build.”*

Apprentice: *“Thank you, master. So that is fate?”*

Mentor: *“Fate? Oh, I thought you said ‘freight’!”*

Tools for Writing XML

- RPG currently provides opcodes XML-SAX and XML-INTO for reading XML. It provides nothing for writing XML
- Cobol has support for writing XML, but it's very limited.
  - –Maybe good enough if you decide on the XML format.
  - –Useless when trying to conform to an existing standard.
  - –No support for name spaces, schemas or even attributes!
- IBM's XML Toolkit supports writing, via the DOM parser.
  - –Very difficult to use.
  - –Documentation is terrible
  - –But, does support attributes, name spaces and schemas!
- Third party tools
- Do it yourself.   *(My recommendation for most circumstances!)*

Complexity of XML

Think about the things that make XML complicated

- maybe multiple elements on a single source record (or maybe not)
- no line breaks in the document (single "record" could be megabytes long -- or maybe not.)
- need to understand name spaces
- document might be in an unexpected character set (or "encoding")

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?><Customer
id="56071" type="retail"><name>Bountiful Grocery, Inc.</name><contact>
Joseph Bachmann</contact><address><street>535 N Wabash Ave</street>
<city>Chicago</city><state>IL</state><postal>60611</postal></address>
</Customer>
```

Think about when these issues might cause problems:

- when reading a document sent by a 3rd-party?   YES!
- when writing an XML document to send out?   NO!

A Problem of Maybes

If you look at the issues on the last slide, you notice a pattern.

- they only cause problems *some of the time.*
- they are only a problem when you can't control how the document has been written.
- when you receive an XML document from a 3rd party, you can't control how they format it. As long as they follow the XML standard, they are doing their part.
- when you generate the XML, you have complete control over these issues. You can format it as you wish.
- therefore, writing XML is several orders of magnitude easier than reading it.

A PF Example (1 of 5)

```
CRTPF FILE(XMLDOC) RCDLEN(1000)
```

```
FCUSTFILE  IF   E           K DISK
FXMLDOC    O   A F 1000       DISK

D Rec             ds          1000

 /free
  exsr WriteHdr;

  setll *start CUSTFILE;
  read CUSTFILE;
  dow not %eof(CUSTFILE);
     exsr WriteXml;
     read CUSTFILE;
  enddo;

  exsr WriteFtr;
  *inlr = *on;
```

It's not too hard to write data in XML format.

This example writes XML to a physical file ("flat file") named XMLDOC.

I'll talk about the problems with this code afterwards.

A PF Example (2 of 5)

```
   begsr WriteHdr;
      rec = '<?xml version="1.0" encoding="utf-8"?>';
      write XMLDOC rec;
      rec = '<CustList>';
      write XMLDOC rec;
   endsr;

   begsr WriteFtr;
      rec = '</CustList>';
      write XMLDOC rec;
   endsr;

   begsr WriteXml;
      rec= '<Customer id="' + %char(CustNo) + '">+
              <name>'     + escape(Name)    + '</name>+
              <contact>' + escape(Contact)+ '</contact>+
              <address>+
                <street>'+ escape(street) + '</street>+
                <city>'   + escape(city)    + '</city>+
                <state>' + escape(state)   + '</state>+
                <postal>'+ escape(postal) + '</postal>+
              </address>+
            </Customer>';
      write XMLDOC rec;
    endsr;
```

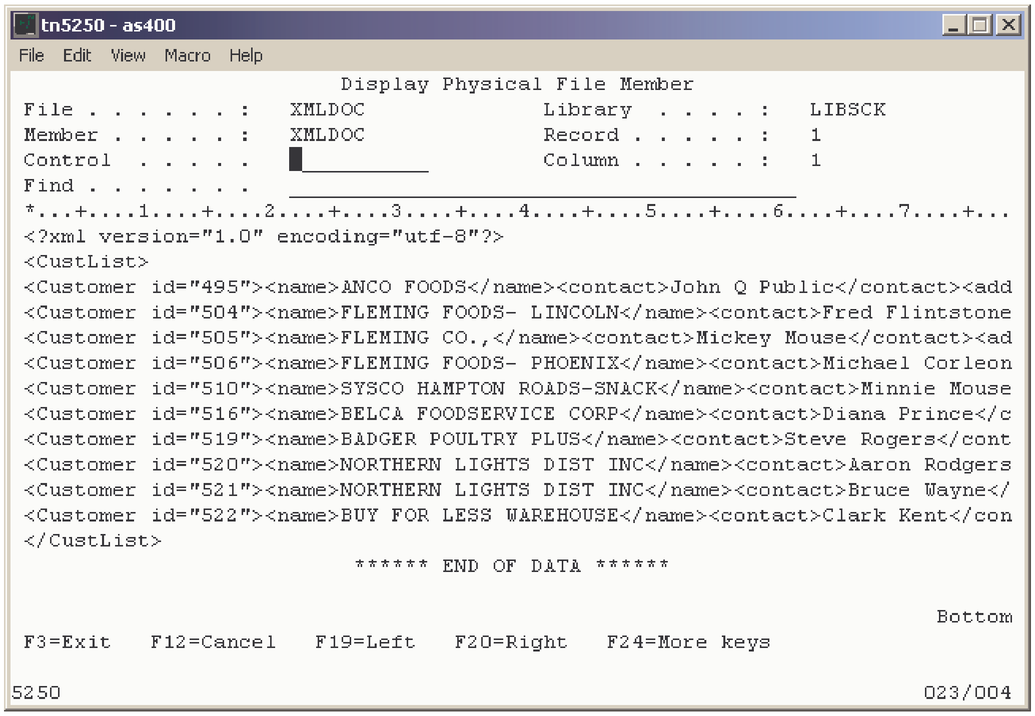
This writes the entire customer to the file with no line breaks – that's okay, since line breaks aren't required in XML.

A PF Example (3 of 5)

```
     P escape          B
     D                 PI           300a   varying
     D  input                       100a   varying const options(*trim)

     D output          s            300a   varying inz('')
     D x               s             10i 0
     D ch              s              1a
      /free
        for x = 1 to %len(input);
          ch = %subst(input:x:1);
          select;
          when ch = '<';
            output += '&lt;';
          when ch = '>';
            output += '&gt;';
          when ch = '&';
            output += '&amp;';
          when ch = '"';
            output += '&quot;';
          when ch = '''';
            output += '&apos;';
          other;
            output += ch;
          endsl;
        endfor;
        return output;
      /end-free
     P                 E
```

This takes care of any special characters in the alphanumeric fields from the CUSTLIST file.

A PF Example (4 of 5)



```
                         Display Physical File Member
File . . . . . . :   XMLDOC              Library  . . . . :   LIBSCK
Member . . . . . :   XMLDOC              Record . . . . . :   1
Control  . . . . .   ___________         Column . . . . . :   1
Find . . . . . . .   ________________________________________
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+...
<?xml version="1.0" encoding="utf-8"?>
<CustList>
<Customer id="495"><name>ANCO FOODS</name><contact>John Q Public</contact><add
<Customer id="504"><name>FLEMING FOODS- LINCOLN</name><contact>Fred Flintstone
<Customer id="505"><name>FLEMING CO.,</name><contact>Mickey Mouse</contact><ad
<Customer id="506"><name>FLEMING FOODS- PHOENIX</name><contact>Michael Corleon
<Customer id="510"><name>SYSCO HAMPTON ROADS-SNACK</name><contact>Minnie Mouse
<Customer id="516"><name>BELCA FOODSERVICE CORP</name><contact>Diana Prince</c
<Customer id="519"><name>BADGER POULTRY PLUS</name><contact>Steve Rogers</cont
<Customer id="520"><name>NORTHERN LIGHTS DIST INC</name><contact>Aaron Rodgers
<Customer id="521"><name>NORTHERN LIGHTS DIST INC</name><contact>Bruce Wayne</
<Customer id="522"><name>BUY FOR LESS WAREHOUSE</name><contact>Clark Kent</con
</CustList>
                          ****** END OF DATA ******

                                                                        Bottom
F3=Exit   F12=Cancel   F19=Left   F20=Right   F24=More keys

5250                                                                   023/004
```

A PF Example (5 of 5)

Ultimately, the IFS data should be put in the IFS in a stream file. (It shouldn't be kept in a physical file.)

- the data is supposed to be UTF-8 Unicode, not EBCDIC
- flat files have fixed-length records, padded with blanks, this makes the files much larger, and therefore less efficient for data interchange.
- XML tools will expect the data to be in the IFS.

It's no problem to convert it, however. (Note: CCSID 1208 is UTF-8)

```
CPYTOSTMF FROMMBR('/qsys.lib/mylib.lib/xmldoc.file/xmldoc.mbr')
          TOSTMF('/tmp/customerList.xml')
          STMFOPT(*ADD) STMFCODPAG(1208) ENDLINFMT(*LF)
```

Using the IFS APIs

The PF example meant writing to one file then copying to another. That's twice as much work for the computer – and slower performance.

Why not write directly to the IFS? Here's a very quick summary of how:

```
/copy IFSIO_H

handle = open( ifs-file-name
             : flags
             : authorities
             : fileccsid
             : pgmccsid );

length = writeA( handle : data : length );

close( handle );
```

IFSIO_H is a copy book available for free from ScottKlement.com

The open() API creates, configures, and opens an IFS file.

The writeA() API writes data to an IFS file. The close() API closes an open file.

An IFS Example (1 of 2)

This program is the same as the PF one, except that I added the /COPY statement, and changed the three subroutines.

```
begsr WriteHdr;
   fd = open( '/tmp/customerList.xml'
            : O_CREAT + O_TRUNC + O_WRONLY + O_INHERITMODE
              + O_CCSID + O_TEXTDATA + O_TEXT_CREAT
            : 0: 1208: 0 );
   if fd = -1;
       // open failed, handle error
   endif;
   rec = '<?xml version="1.0" encoding="utf-8"?>'
       + '<CustList>';
   callp writeA(fd: rec: %len(rec));
endsr;

begsr WriteFtr;
   rec = '</CustList>';
   callp writeA(fd: rec: %len(rec));
   callp close(fd);
endsr;
```

An IFS Example (2 of 2)

This program is the same as the PF one, except for the three subroutines. I've modified them to use the IFS APIs instead.

```
        begsr WriteXml;
          rec= '<Customer id="' + %char(CustNo) + '">+
                  <name>'    + escape(Name)    + '</name>+
                  <contact>' + escape(Contact)+ '</contact>+
                  <address>+
                    <street>'+ escape(street) + '</street>+
                    <city>'  + escape(city)   + '</city>+
                    <state>' + escape(state)  + '</state>+
                    <postal>'+ escape(postal) + '</postal>+
                  </address>+
                </Customer>';
          callp writeA(fd: rec: %len(rec));
        endsr;
```

Now my XML is written straight to the IFS. No middle-man!

Do You Want XML in your RPG?

As your XML documents get more complex, coding the XML tags inside your RPG code can be a challenge. Many RPG programmers have asked me for a better solution.

Why not CGIDEV2?

- As you've noticed, XML looks a lot like HTML.
- CGIDEV2 is designed for HTML, but works with XML as well.
- `WrtHtmlStmf()` routine lets you write to the IFS instead of sending your output via HTTP.

CGIDEV2 provides a very simple way of writing XML from an RPG application.

- A "template" for the XML is put in an IFS file.
- You divide your template into "sections" or "chunks" to be written at one time.
- You specify fill-in "variables" that will be populated from your RPG code.

CGIDEV2 Template

This program is the same as the PF one, except that I added the /COPY statement, and changed the three subroutines.

```
/$Header
<?xml version="1.0" encoding="iso-8859-1"?>
<CustList>
/$Customer
   <Customer id="/%CustNo%/">
     <name>/%Name%/</name>
     <contact>/%Contact%/</contact>
     <address>
       <street>/%street%/</street>
       <city>/%city%/</city>
       <state>/%state%/</state>
       <postal>/%postal%/</postal>
     </address>
   </Customer>
/$Footer
</CustList>
```

At present, CGIDEV2 doesn't support Unicode (without modifying the source.)

So we'll use ISO-8859-1 (ASCII) instead.

ISO-8859-1 is CCSID 819.

CGIDEV2 RPG Example

```
     FCUSTFILE  IF   E           K DISK
      . . .

       getHtmlIfsMult( '/xmldemo/CustListTemplate.xml' );
       clrHtmlBuffer();
       wrtsection('Header');

       setll *start CUSTFILE;
       read CUSTFILE;

       dow not %eof(CUSTFILE);

          updHtmlVar('Custno'  : %char(CustNo)  );
          updHtmlVar('Name'    : escape(Name)   );
          updHtmlVar('Contact' : escape(Contact));
          updHtmlVar('Street'  : escape(Street) );
          updHtmlVar('City'    : escape(City)   );
          updHtmlVar('State'   : escape(state)  );
          updHtmlVar('Postal'  : escape(Postal) );
          wrtsection('Customer');

          read CUSTFILE;
       enddo;

       wrtsection('Footer');
       WrtHtmlToStmf('/tmp/custList_cgidev.xml': 819);

       *inlr = *on;
      /end-free
```

getHtmlIfsMult() loads the template from the last slide.

wrtsection() writes one section (chunk) of HTML.

updHtmlVar() replaces variable data with data from this RPG program (in this example,fields from the CUSTLIST file.)

Writing XML – Final Thoughts

It's much easier to write than read

Writing to a PF is familiar, and not hard to do

Writing to the IFS might require a learning curve, but performs better and works better.

CGIDEV2 works better when the XML structure gets complex, because you don't have to worry about RPG's quoting, or blocking up your code with complicated XML.

None of these options provide schema validation!

- but you will test your code, won't you?
- run a few samples through a separate schema validator as part of the testing process.
- is it important to validate every time you write XML?

This Presentation

You can download a PDF copy of this presentation from:

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

The sample code shown in this talk can also be downloaded from the preceding link.

Thank you!

Reading XML From RPG

Presented by

Scott Klement

http://www.scottklement.com

```
Type reply (if required), press Enter.
  User error. Replace user to continue (C G).
    Reply . . :   G
```

Tools for Reading XML

- With V5R3, a SAX parser was added to COBOL for parsing XML.
- In V5R4, RPG added two op-codes for XML support
  - XML-SAX: Standard SAX XML parser
  - XML-INTO: Uses SAX under the covers, but takes care of mapping the data to your data structure for you.
- IBM's XML Toolkit
  - Provides SAX parser
  - Provides DOM parser
  - Very hard to use – documentation is terrible.
- Expat, open source parser
  - HTTPAPI provides a simplified interface to Expat
- Third-party tools.

XML as a Data Structure

You might think of an XML document as a data structure. Compare this:

```
<address>
  <street>123 Sesame St.</street>
  <city>New York</city>
  <state>NY</state>
  <postal>10012</postal>
</address>
```

To this:

```
 D address         ds                   qualified
 D   street                      30a
 D   city                        20a
 D   state                        2a
 D   postal                      10a
```

They both describe data with the same structure.

Path to XML Data

You might think of an XML document as a data structure. Compare this:

```
<address>
  <street>123 Sesame St.</street>
  <city>New York</city>
</address>
```

Think about how you "get to" the values. Which XML elements are they stored inside?

To get to "123 Sesame St.":

- go through address / street

To get to "New York"

- go through address / city
- but not through "street". The city tag is "outside" of street.

This is the "path" to the data.

The Gist of XML-INTO

```
D my_data         ds                  qualified
D   street                         30a
D   city                           20a
D   state                           2a
D   postal                         10a

     my_doc = '<address>+
                 <street>123 Sesame St.</street>+
                 <city>New York</city>+
                 <state>NY</state>+
                 <postal>10012</postal>+
               </address>';

      xml-into my_data
               %xml(my_doc: 'path=address case=any');
```

`path=address` **tells XML-INTO that the sub-elements of the** `<address>` **element should be mapped to** `my_data`**'s subfields.**

`case=any` **tells XML-INTO to ignore upper/lower case when mapping into RPG field names.**

XML-INTO maps the XML data into the data structure. When this code completes:

- `my_data.street = '123 Sesame St.'`
- `my_data.city = 'New York'`
- `my_data.state = 'NY';`
- `my_data.postal = '10012'`

XML-INTO Syntax Overview

```
XML-INTO{(EH)} receiver %XML(xmlDoc {: options });
```

Opcode extenders:

- E = instead of a halt, turn on %ERROR indicator.
- H = half-adjust if needed when assigning numerics.
  (in case there are numeric values in the data structure / receiver)

receiver = variable to map XML data into. Can be a standalone variable, or (more likely) a data structure.

%XML(xmlDoc : options) = BIF pertaining to the XML document

- ***xmlDoc*** identifies either a character string or IFS path of XML document
- ***options*** is for options that control how the XML is processed

```
opts = 'path=address case=any';
xml-into my_data %xml( my_doc : opts );
```

More About %XML BIF

```
%XML(xmlDoc {: options });
```

xmlDoc represents an RPG variable

- by default, this is an RPG variable containing the XML data.
- if the `doc=file` option has been set, xmlDoc represents an IFS path name to the XML document to read. (RPG will open/read it from the IFS for you.)

options controls the XML processing

- character string or literal. (can be changed at run-time, if needed.)
- format is keyword=value keyword=value, etc.
- no spaces between keywords and values
- space denotes the end of one keyword, and start of the next.

%XML BIF Options (1 of 3)

path **=** identifies the part of the XML document to process

- in my example, 'address' is the outermost XML element to process
- by default, path is unspecified. In that case, the variable name has to match the outermost XML element in the document. (if I had named my data structure 'address', path would be unnecessary.)

doc controls the type of document storage

- doc=string means the XML will be read from a variable (default)
- doc=file means the XML will be read from an IFS path name

case controls the way XML tag names are matched to RPG field names

Note: I always use case=any. And, in my opinion, this option isn't very useful.

- case=lower means lowercase XML tags match RPG names (default)
- case=upper means uppercase XML tags match RPG names
- case=any means that matches are case-insensitive.

%XML BIF Options (2 of 3)

trim trims extra blanks, tabs and newlines from the XML data

- trim=all causes tabs, newlines, and blanks to be trimmed from the data. *Not just the leading and trailing ones!* They are trimmed from the middle, too, resulting in just a single blank between words. (default)
- trim=none leaves any blanks, tabs or newlines unchanged.

ccsid lets you override the character set of the document.

- ccsid=best means that RPG will use the file's CCSID if available, or the job's CCSID if not. (default)
- ccsid=job means that RPG will treat the XML as being in the job's (EBCDIC) CCSID
- ccsid=ucs2 means that RPG will treat the XML as being in UCS-2 Unicode.
- Note: Frustratingly, RPG doesn't have an option to calculate the CCSID from the `<?xml?>` processing directive. It always relies on one of the three options, above.

%XML BIF Options (3 of 3)

allowMissing specifies how RPG reacts when the XML document is missing a tag or attribute that you've defined in your variable.

- allowmissing=no means that everything you've defined in your variable must be present in the XML document. Otherwise, an error is triggered.
- allowmissing=yes means that you can have fields/elements in your variable that do not exist in the XML document.

allowExtra specifies how RPG reacts when an XML document has "extra" elements or attributes that you haven't specified in your variable.

- allowextra=no means that there can't be any "extra" elements in the XML data. Your data structure must have space for all of them. (default)
- allowextra=yes means that extra XML elements or attributes will be silently ignored.

Note: In my opinion, these options were misguided. allowextra=yes should always be used, because organizations have the right to extend their XML documents.

And for all practical purposes, allowMissing must be yes, otherwise you can't use arrays for repeating elements if the size of the list varies.

Nested Structures (1 of 3)

```
D address_t       ds                   qualified
D   street                      30a
D   city                        20a
D   state                        2a
D   postal                      10a

D Customer        ds                   qualified
D   id                           5p 0
D   type                        10a
D   name                        30a
D   contact                     30a
D   address                            likeds(address_t)

D file            s            100a    varying
D opts            s            100a    varying
 /free
  file = '/xmldemo/simple.xml';
  opts = 'doc=file case=any';
  xml-into Customer %xml(file: opts);
```

A qualified data structure can contain a subfield defined with LIKEDS. That subfield will be be a data structure, nested within another data structure!

Nested Structures (2 of 3)

XML data can be nested. It doesn't always correspond to a single/simple data structure. For example: (This is still much simpler than most real-world XML!)

```
<Customer id="56071" type="retail">
   <name>Bountiful Grocery, Inc.</name>
   <contact>Joseph Bachmann</contact>
   <address>
     <street>535 N Wabash Ave</street>
     <city>Chicago</city>
     <state>IL</state>
     <postal>60611</postal>
   </address>
</Customer>
```

The <address> element makes a nice, normal, data structure. But the <Customer> element contains address -- it's like a data structure within a data structure!

Nested Structures (3 of 3)

After the code from the previous slide has run, the following subfields will be filled in, as follows:

```
CUSTOMER.ID             = 56071.
CUSTOMER.TYPE           = 'retail    '
CUSTOMER.NAME           = 'Bountiful Grocery, Inc.        '
CUSTOMER.CONTACT        = 'Joseph Bachmann                '
CUSTOMER.ADDRESS.STREET = '535 N Wabash Ave               '
CUSTOMER.ADDRESS.CITY   = 'Chicago             '
CUSTOMER.ADDRESS.STATE  = 'IL'
CUSTOMER.ADDRESS.POSTAL = '60611     '
```

You refer to nested fields in your code just like any other variable (except that it contains the names of the structures as a prefix):

```
if Customer.Address.State = 'IL'
    and Customer.Type = 'retail';
   // handle business rules that apply to Illinois retail
   // customers.
endif;
```

Repeated XML Elements (1 of 3)

To demonstrate a list of repeating XML elements, I'm going to use the following sample XML:

```
<?xml version="1.0" encoding="utf-8"?>
<CustList>
 <Customer id="495"><name>ANCO FOODS &amp; FRIENDS</name><contact>John
 <Customer id="504"><name>FLEMING FOODS- LINCOLN</name><contact>Fred Fl
 <Customer id="505"><name>FLEMING CO.,</name><contact>Mickey Mouse</con
 <Customer id="506"><name>FLEMING FOODS- PHOENIX</name><contact>Michael
 <Customer id="510"><name>SYSCO HAMPTON ROADS-SNACK</name><contact>Minn
 <Customer id="516"><name>BELCA FOODSERVICE CORP</name><contact>Diana P
 <Customer id="519"><name>BADGER POULTRY PLUS</name><contact>Steve Roge
 <Customer id="520"><name>NORTHERN LIGHTS DIST INC</name><contact>Aaron
 <Customer id="521"><name>NORTHERN LIGHTS DIST INC</name><contact>Bruce
 <Customer id="522"><name>BUY FOR LESS WAREHOUSE</name><contact>Clark K
</CustList>
```

Note that I could not fit all XML elements across the width of the slide. But, all of the subfields from the previous examples are included for each customer in the list.

Repeated XML Elements (2 of 3)

```
D address_t       ds                    qualified
D   street                        30a
D   city                          20a
D   state                          2a
D   postal                        10a

D Customer_t      ds                    qualified
D   id                             5p 0 inz
D   type                          10a
D   name                          30a
D   contact                       30a
D   address                             likeds(address_t)

D CustList        ds                    qualified
D   Customer                            likeds(Customer_t)
D                                       inz(*likeds)
D                                       dim(400)
```

Adding the DIM() to my data structure allows it to repeat, as an array:

One problem is that I don't know how many customers will be in the customer list. When I code DIM(400), I had to pick the largest number I could ever handle. Most of the time, my XML document will have fewer than 400…

Repeated XML Elements (3 of 3)

```
file = '/home/klemscot/xmldemo/customerList.xml';
opts = 'doc=file case=any allowmissing=yes';

xml-into CustList %xml(file: opts);

for x = 1 to %elem(CustList.customer);
   if CustList.Customer(x).id = 0;
      leave;
   endif;
   dsply CustList.Customer(x).name;
endfor;
```

Since my XML document will have fewer than 400 `<Customer>` tags, I *must* specify allowmissing=yes, otherwise I'll get an error when XML-INTO runs.

In the original V5R4 implementation of XML-INTO, it was hard to determine how many array entries were found.

- there's a field in the PSDS for this, but it doesn't work if your XML document contains more than one array (most do!)
- my workaround is to set the elements to blanks/zeros using INZ(*LIKEDS) and end the loop when a zero/blank value is found in a required element.

XML-INTO With Too Much Data

```
D CustList        ds                    qualified
D   Customer                            likeds(Customer_t)
D                                       inz(*likeds)
D                                       dim(400)
```

One problem with XML-INTO (especially at v5r4) is that it's easy to create variables that are too big.

- My **CustList** DS is already 54000 bytes long with DIM(400)
- At V5R4, it can only be 65535 or smaller. So DIM(500) would fail.
- In 6.1 and 7.1, CustList could be as large as 16 MB. But with a complex XML document that has nested loops, that is also easily exceeded.

But I don't really need to load my whole customer file into memory at once!

In many cases, I don't necessarily need 500 (or more) iterations of the `<Customer>` element, I only need one at a time.

That's what %HANDLER is for.

XML-INTO Under the Covers

We never see the code that runs behind the XML-INTO opcode, but under the covers, there's computer code, written by IBM.

It works something like this (pseudocode)

```
dow more data;

   read next element or attribute;

   load into data structure;

   if DS Completely Filled In = Yes;
      add 1 to array index;
   endif;
enddo;
```

The problem is that "add 1 to array index" part. If there's a lot of data, it could eventually get larger than RPG can store.

What's needed is to let the program process the data immediately, rather than load more and more into the array!

XML-INTO Calling a Handler

Suppose IBM lets the code work like this, instead:

```
dow more data;

   read next element or attribute;

   load into data structure;

   if DS Completely Filled In = Yes;
      callp ScottsHandler( the data structure );
   endif;
enddo;
```

Again… we never see the internals of the XML-INTO opcode. But somewhere in there, it's looping through the XML document.

Wouldn't it be great if that loop could call my "custom routine" when it's time to process one 'record' of data?

That's what %HANDLER() is for.

XML-INTO Calling a Handler

XML-INTO lets you tell it a routine to be called from it's processing loop by using the %HANDLER BIF in place of the receiver variable.

```
%HANDLER( Subprocedure-to-call : Parameter );
```

Notes:

- When specifying %HANDLER, the path= option is required.
- A handler receives it's data structure in the 2nd parameter. This can be an array if you want RPG to pass you a group of records to process at once.

Parameter ('Communication Area')

- The parameter (oddly named 'communication area') instructs RPG to pass any one variable to your handler.
- This is for your benefit (RPG doesn't use the variable for anything) to let you communicate data from the routine calling XML-INTO to the handler.

Handler Example (1 of 4)

```
H DFTACTGRP(*NO)

D address_t       ds                  qualified
D   street                      30a
D   city                        20a
D   state                        2a
D   postal                      10a

D Customer_t      ds                  qualified
D   id                           5p 0 inz
D   type                        10a
D   name                        30a
D   contact                     30a
D   address                           likeds(address_t)

D MyHandler       PR            10i 0
D   loaded                      10i 0
D   cust                              likeds(Customer_t) dim(5)
D                                     const
D   count                       10i 0 value
```

Handler Example (2 of 4)

```
D file            s            100a   varying
D opts            s            100a   varying
D loaded          s             10i 0

/free
  file = '/home/klemscot/xmldemo/customerList.xml';
  opts = 'doc=file case=any allowmissing=yes +
          path=CustList/Customer';

  xml-into %Handler( MyHandler : loaded ) %xml(file: opts);

  dsply ('I loaded ' + %char(loaded) + ' elements.');

  *inlr = *on;
/end-free
```

Now I've told RPG:

- which subprocedure (MyHandler) to call in it's loop
- to pass 'loaded' as a parameter to that procedure.
- RPG will also pass the data structure
- and a count of elements loaded into the data structure

Handler Example (3 of 4)

```
P MyHandler       B
D                 PI            10i 0
D   loaded                      10i 0
D   cust                              likeds(Customer_t) dim(5)
D                                     const
D   count                       10i 0 value

D x               s             10i 0
 /free
   for x = 1 to count;
      dsply cust(x).contact;
      CustNo  = cust(x).id;
      Name    = cust(x).name;
      Contact = cust(x).contact;
      Street  = cust(x).address.street;
      City    = cust(x).address.city;
      State   = cust(x).address.state;
      Postal  = cust(x).address.postal;
      write CUSTFILEF;
   endfor;
   loaded += count;
   return 0;
 /end-free
P                 E
```

Handler Example (4 of 4)

```
> call MYPGM
  DSPLY  John Q Public
  DSPLY  Fred Flintstone
  DSPLY  Mickey Mouse
  DSPLY  Michael Corleone
  DSPLY  Minnie Mouse
  DSPLY  Diana Prince
  DSPLY  Steve Rogers
  DSPLY  Aaron Rodgers
  DSPLY  Bruce Wayne
  DSPLY  Clark Kent
  DSPLY  I handled 10 elements.
```

Plus, of course, my **`CUSTFILE`** file has been populated with records loaded from the XML document.

XML-INTO PTFs (1 of 4)

```
<example>
  <author type="freelance">Scott Klement</author>
</example>
```

Problem: If you were faced with an issue like this, how would you define the data structure?

You can't. An RPG data structure can't have both a subfield (type) and store regular data ("Scott Klement").

The PTF enables a new option named "datasubf". With datasubf, you define the name of a data structure subfield that should contain character data. For example:

```
  XML-INTO example  %XML( stmf : 'doc=file datasubf=value' );
```

```
 D author_t        ds                    qualified
 D   type                          15a
 D   value                         30a

 D example         ds                    qualified
 D   author                              likeds(author_t)
```

XML-INTO PTFs (2 of 4)

Problem: how can you get the count of multiple arrays in the same document?

- employee array
- dependent array

```
<example>
  <employee name="Bob">
    <dependents>
      <dependent name="Cindy" rel="spouse"/>
      <dependent name="Alice" rel="child"/>
      <dependent name="Jason" rel="child"/>
    </dependents>
  </employee>
  <employee name="Carol">
    <dependents>
      <dependent name="John" rel="spouse"/>
      <dependent name="Dean" rel="child"/>
    </dependents>
  </employee>
</example>
```

XML-INTO PTFs (3 of 4)

Solution: new countprefix option. Enables RPG to put the count of elements in an array into a data structure subfield. Any subfield can be counted.

```
    XML-INTO example %XML(doc: 'doc=file countprefix=cnt');

D dependent_t     ds                    qualified
D   name                           30a
D   rel                            10a

D dependents_t    ds                    qualified
D   cntDependent                   10i 0
D   dependent                           likeds(dependent_t)
D                                       dim(50)

D employee_t      ds                    qualified
D   name                           30a
D   dependents                          likeds(dependents_t)

D example         ds                    qualified
D   employee                            likeds(employee_t)
D   cntEmployee                   10i 0
```

XML-INTO PTFs (4 of 4)

Problem: XML element names allow different characters than RPG variables.

```
<retail-invoice> ... some data here ... </retail-invoice>
-or-
<tns:doc xmlns:tns="http://example.com/example">
   <tns:something />
</tns:doc>
```

Prior to the PTFs (or if you're using V5R4) you'd have to use XML-SAX for these documents. XML-INTO can't work, because you can't put a dash or colon character in an RPG variable name. New options let you replace bad characters with underscores.

PTF adds support for translating these special characters into underscores via the new case=convert, ns and nsprefix options.

Details are here for datasubf and countprefix:
http://systeminetwork.com/article/ptfs-version-61-enhance-rpgs-xml

And for case=convert, ns and nsprefix:
http://systeminetwork.com/article/xml-namespace-support-added-rpgs-xml

XML-SAX Concepts

SAX stands for "Simple API for XML"

- simpler than DOM
- but perhaps not as simple as we'd like it to be!
- XML-INTO uses SAX under the covers, but adds extra features in order to make it simpler yet.
- Sometimes the added flexibility of SAX is useful, however.

When you use XML-SAX:

- it loads the file from disk (doc=file) or a variable (doc=string) just as XML-INTO would.
- Output is always done with %HANDLER.
- *You* are responsible for mapping the data into your data structure
  - or file
  - or array
  - or subfile
  - or wherever you want to put the data!

How SAX Works

SAX works by scanning your XML document, character-by-character, looking for the special XML characters so that it can recognize:

- the opening tag of an element
- the closing tag of an element
- attributes and their values
- entity references.
- character data between tags.

When it recognizes any of the above:

- it gets very excited :-)
- it wants to tell you about it!!
- it calls your %HANDLER routine, and tells you what it found.
- after calling your handler, it continues looking for the next event.

SAX Scanning For Events

Here's an example of the events:

```
<CustList>
   <Customer id="1234" type="retail">
     <name>Fred's Pig Sprinklers, Inc.</name>
     <contact>Fred</contact>
     <address>
       <street>123 Main St.</street>
       <city>Cleveland</city>
       <state>OH</state>
       <postal>44145</postal>
     </address>
   </Customer>
   <Customer>
   . .
   </Customer>
   . . More Customers Here . .
</CustList>
```

Start events in red

Character data events in black

End events in blue

How XML-SAX Notifies You

whenever XML-SAX finds an event, it calls your %HANDLER. The handler must always have the following parameters:

```
D your-procedure  PR              10i 0
D   commArea                      -any-
D   event                         10i 0 value
D   string                          *   value
D   stringLen                     20i 0 value
D   exceptionId                   10i 0 value
```

- ***commArea*** = user defined parameter (same as XML-INTO's)
- ***event*** = number indicating the event that occurred
- ***string*** = pointer to data related to the event
  - for start/end events this is an attribute/element name
  - for character data, this is the data
- ***stringLen*** = length of string parameer
- ***exceptionId*** = only used for exception events. Has the XML parser's error code.

Event Values

When your procedure is called, the event parameter is a number that indicates which event has occurred. RPG provides named constants for each possible event. Here are some of them:

*XML_START_DOCUMENT	start of entire XML document
*XML_START_ELEMENT	opening tag found (start of an element)
*XML_ATTR_NAME	attribute name found (start of an attribute)
*XML_CHARS	character data found (between tags)
*XML_ATTR_CHARS	character data found (in attribute value)
*XML_PREDEF_REF	predefined entity reference
*XML_UCS2_REF	predefined UCS-2 entity reference
*XML_UNKNOWN_REF	another reference (not predefined) found
*XML_END_DOCUMENT	end of entire XML document
*XML_END_ELEMENT	closing tag found (end of an element)
*XML_END_ATTR	end of an attribute value

Keep Track of Tag Path

When using SAX, you are going to need to keep track of the XML element you're currently working with.

You'll need this information to map your value to a field, later.

This is best done with a stack -- sort of like piling up boxes as you load them for moving your office.

Tracking Position

The way I keep track of where I am in the document is by implementing a stack. Each time the start element handler is called, it pushes a new item onto the stack. Each time the end element handler is called, it pops the top item off of the stack.

```
   <CustList>
      <Customer id="1234" type="retail">
        <name>Fred's Pig Sprinklers, Inc.</name>
        <address>
          <street>123 Main St.</street>
          <city>Cleveland
           . . . and so on . . .
```

/CustList/Customer/address/city

/CustList/Customer/address

/CustList/Customer

/CustList

Boxes in Code

To keep track of a stack of boxes in code, do the following:

- have arrays to keep track of the stack

```
      D MAX_DEPTH       c                   32
      D depth           s             10i 0 inz(0) static
      D stackname       s          65535a   varying inz('')
      D                                     dim(MAX_DEPTH)
      D stackval        s          65535a   varying inz('')
      D                                     dim(MAX_DEPTH)
```

- when XML-SAX tells you it found the start of a new element
    - add 1 to the array index
    - store the path to the new element into stackname array.
    - clear out the value.
- when XML-SAX tells you that character data has been found
    - save it to the current array stackval(x) array.
- when XML-SAX tells you that it found the end of an element
    - map the value into a data structure or record
    - subtract one from the array index

XML-SAX Example (1 of 6)

```
H DFTACTGRP(*NO)

FCUSTFILE  O  A E             K DISK

D BobsYourUncle   PR              10i 0
D   loaded                        10i 0
D   event                         10i 0 value
D   string                          *   value
D   stringLen                     20i 0 value
D   exceptionId                   10i 0 value

D CustRec         ds                    likerec(CUSTFILEF:*OUTPUT)
D loaded          s               10i 0

 /free

  xml-sax %handler(BobsYourUncle: loaded)
          %XML('xmldemo/customerList.xml': 'doc=file');
  dsply ('I handled ' + %char(loaded) + ' elements.');

  *inlr = *on;

 /end-free
```

XML-SAX Example (2 of 6)

```
P BobsYourUncle   B
D                 PI              10i 0
D   loaded                        10i 0
D   event                         10i 0 value
D   string                          *   value
D   stringLen                     20i 0 value
D   exceptionId                   10i 0 value

D value           s            65535a   based(string)
D ucs2val         s            16383c   based(string)

D MAX_DEPTH       c                     32
D depth           s               10i 0 inz(0) static
D stackname       s            65535a   varying inz('')
D                                       dim(MAX_DEPTH)
D                                       static
D stackval        s            65535a   varying inz('')
D                                       dim(MAX_DEPTH)
D                                       static
```

XML-SAX Example (3 of 6)

```
 /free
   select;
   when event = *XML_START_ELEMENT;
      depth += 1;
      if depth = 1;
         stackname(depth) = %subst(value:1:stringLen);
      else;
         stackname(depth) = stackname(depth-1) + '/'
                          + %subst(value:1:stringLen);
      endif;
      stackval(depth)  = '';

      callp startEvent( loaded : stackname(depth) );

   when event = *XML_ATTR_NAME;
      depth += 1;
      stackname(depth) = stackname(depth-1) + '/@'
                       + %subst(value:1:stringLen);
      stackval(depth) = '';

      callp startEvent( loaded: stackname(depth) );
```

XML-SAX Example (4 of 6)

```
   when event = *XML_END_ELEMENT
     or event = *XML_END_ATTR;

      callp endEvent( loaded
                    : stackname(depth)
                    : stackval(depth) );
      depth -= 1;

   when event = *XML_CHARS
     or event = *XML_PREDEF_REF
     or event = *XML_ATTR_CHARS
     or event = *XML_ATTR_PREDEF_REF;
      stackval(depth) += %subst(value:1:stringLen);

   when event = *XML_UCS2_REF
     or event = *XML_ATTR_UCS2_REF;
      stackval(depth) += %char(%subst( ucs2val : 1
                                     : %div(stringLen:2) ));
   endsl;

   return 0;
 /end-free
P                  E
```

XML-SAX Example (5 of 6)

```
P startEvent      B
D                 PI
D   loaded                        10i 0
D   path                       65535a    varying const
 /free
    if path = 'CustList/Customer';
       loaded += 1;
       clear CustRec;
    endif;
 /end-free
P                 E
```

XML-SAX Example (6 of 6)

```
P endEvent        B
D                 PI
D   loaded                        10i 0
D   path                       65535a    varying const
D   value                      65535a    varying const
 /free
    select;
    when path = 'CustList/Customer/@id';
       CustRec.custno = %int(value);
    when path = 'CustList/Customer/name';
       CustRec.name = value;
    when path = 'CustList/Customer/contact';
       CustRec.contact = value;
    when path = 'CustList/Customer/address/street';
       CustRec.street = value;
    when path = 'CustList/Customer/address/city';
       CustRec.city = value;
    when path = 'CustList/Customer/address/state';
       CustRec.state = value;
    when path = 'CustList/Customer/address/postal';
       CustRec.postal = value;
    when path = 'CustList/Customer';
       write CUSTFILEF CustRec;
    endsl;
 /end-free
P                 E
```

This Presentation

You can download a PDF copy of this presentation from:

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

The sample code shown in this talk can also be downloaded from the preceding link.

Thank you!

Web Services Basics

Presented by

Scott Klement

http://www.scottklement.com

"Programming is like sex, one mistake and you have to support it for the rest of your life."

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!

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)

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....

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.

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

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?

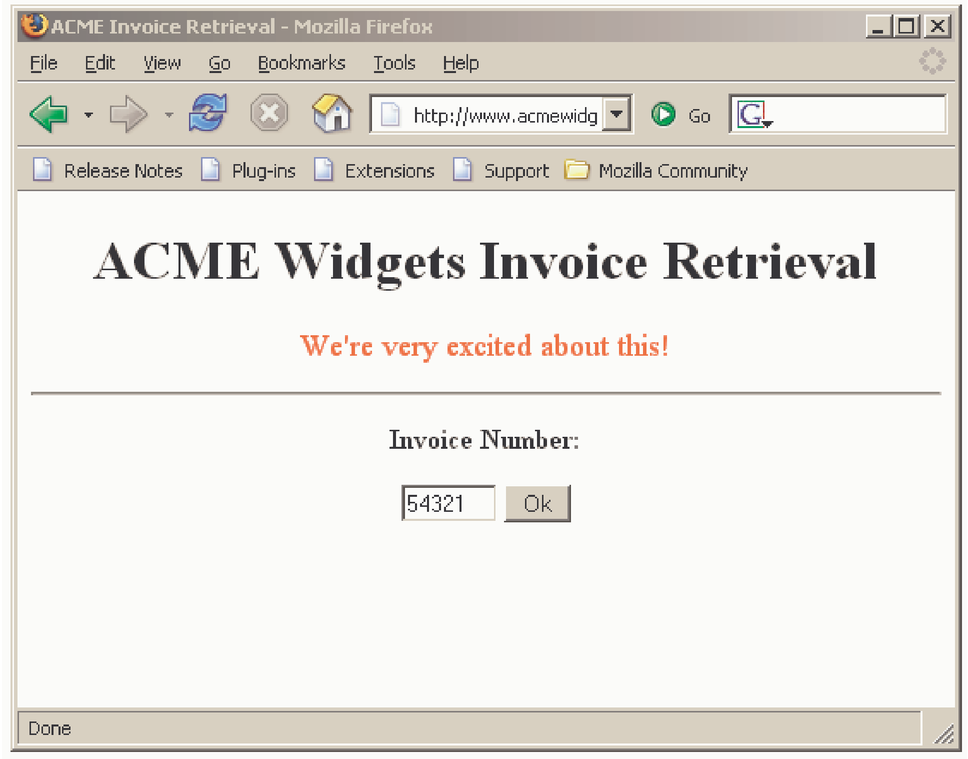
Well, let's start by examining how a simple web application might work…

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

Web Enabled Invoice


ACME Invoice Retrieval - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.acmewidg
Go
Release Notes
Plug-ins
Extensions
Support
Mozilla Community

ACME Widgets Invoice Retrieval

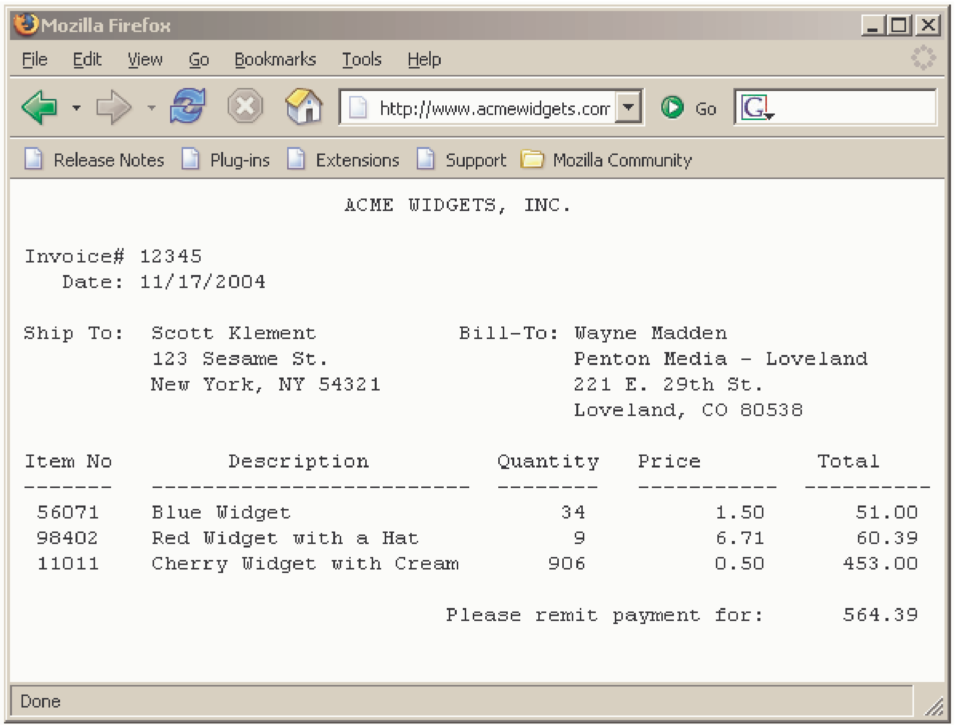
We're very excited about this!

Invoice Number:

54321 Ok

Done

Web Enabled Invoice


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.acmewidgets.com
Go
Release Notes Plug-ins Extensions Support Mozilla Community
Done

```
                         ACME WIDGETS, INC.

Invoice# 12345
   Date: 11/17/2004

Ship To:  Scott Klement          Bill-To: Wayne Madden
          123 Sesame St.                  Penton Media - Loveland
          New York, NY 54321              221 E. 29th St.
                                          Loveland, CO 80538

Item No         Description           Quantity   Price         Total
-------   -------------------------   --------   -----------  ----------
 56071    Blue Widget                      34          1.50       51.00
 98402    Red Widget with a Hat             9          6.71       60.39
 11011    Cherry Widget with Cream        906          0.50      453.00

                                 Please remit payment for:       564.39
```

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

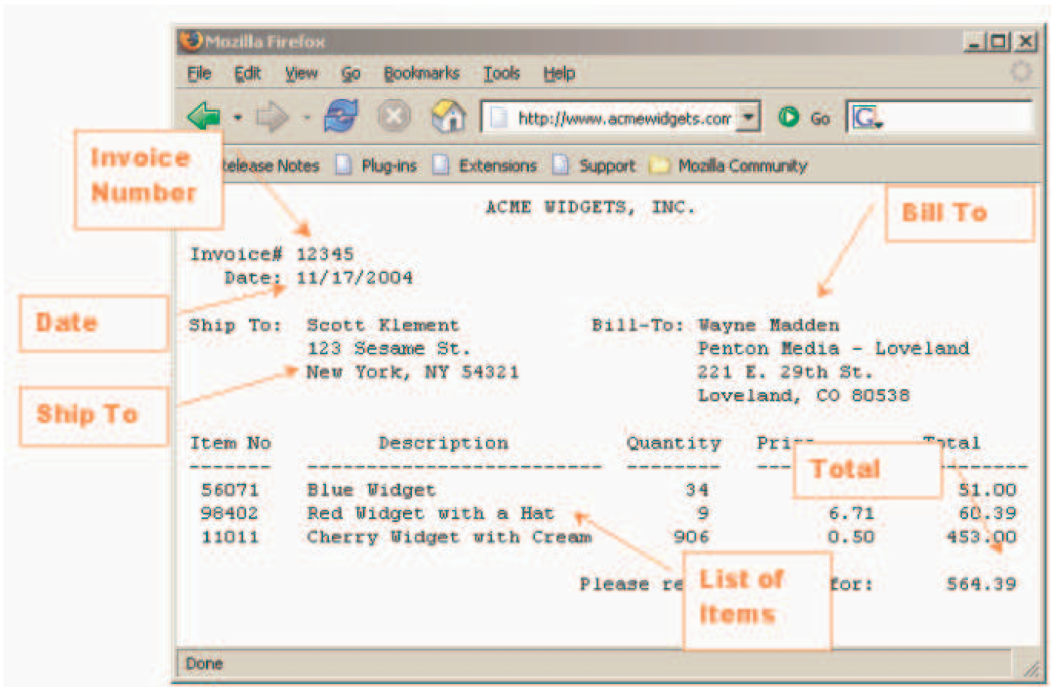
Need to Know "What"

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."

"Marked Up" Data



```
                    ACME WIDGETS, INC.

Invoice# 12345
   Date: 11/17/2004

Ship To:  Scott Klement           Bill-To: Wayne Madden
          123 Sesame St.                   Penton Media - Loveland
          New York, NY 54321               221 E. 29th St.
                                           Loveland, CO 80538

Item No         Description            Quantity  Pri         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           for:        564.39
```

"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>
```

"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>
```

XML Is Only One Option

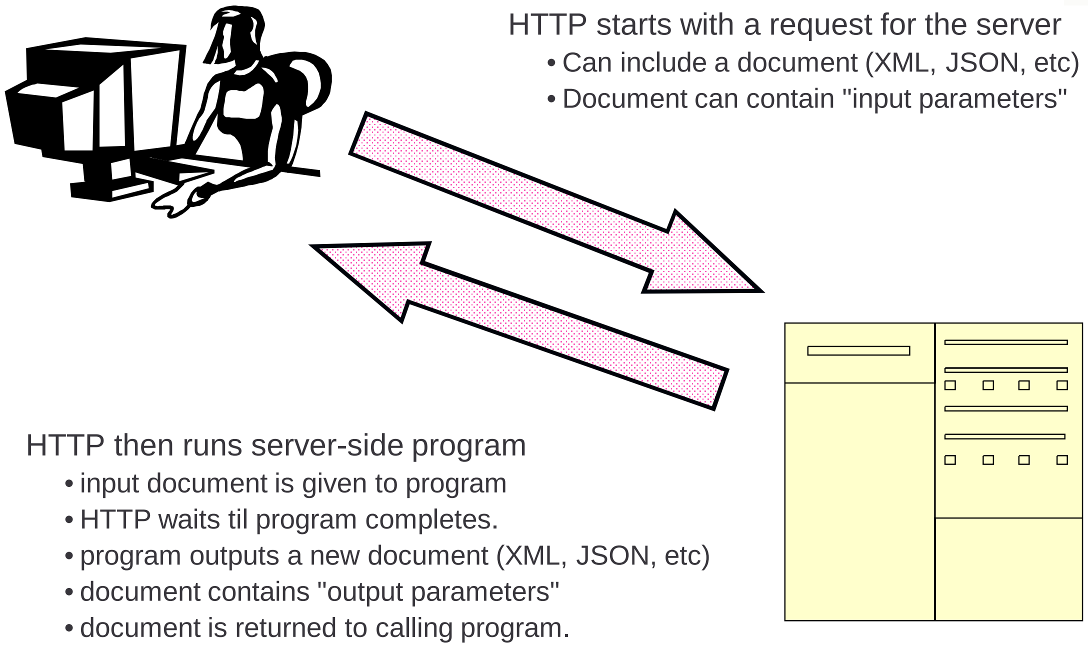
Most web services use XML

- 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)

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.

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 parmeters
  - 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/replace the customer
- DELETE -- delete the customer

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>
```

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

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…

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.

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??!

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)

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.)

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!

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>
```

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>
```

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 interchangable)
- 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

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>
```

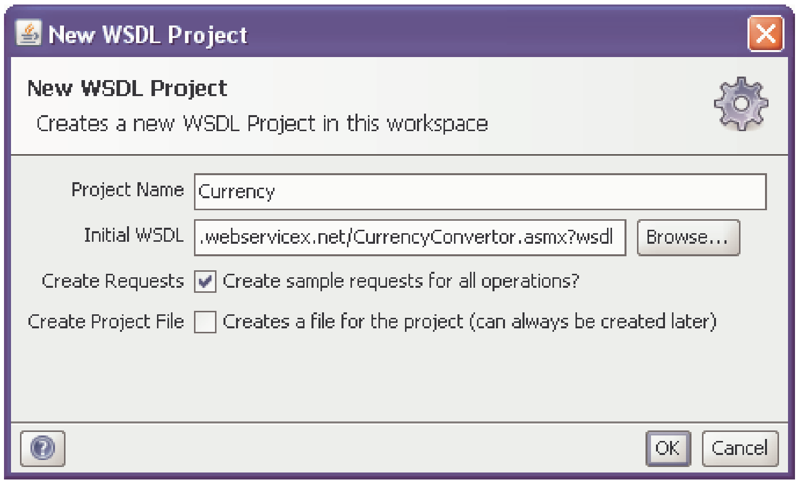
Output Message

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

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 WDSL document. http://www.soapui.org

Click ***File / New WSDL Project***



New WSDL Project
Creates a new WSDL Project in this workspace

Project Name: Currency
Initial WSDL: .webservicex.net/CurrencyConvertor.asmx?wsdl
Create Requests: [✓] Create sample requests for all operations?
Create Project File: [ ] Creates a file for the project (can always be created later)

PROJECT NAME

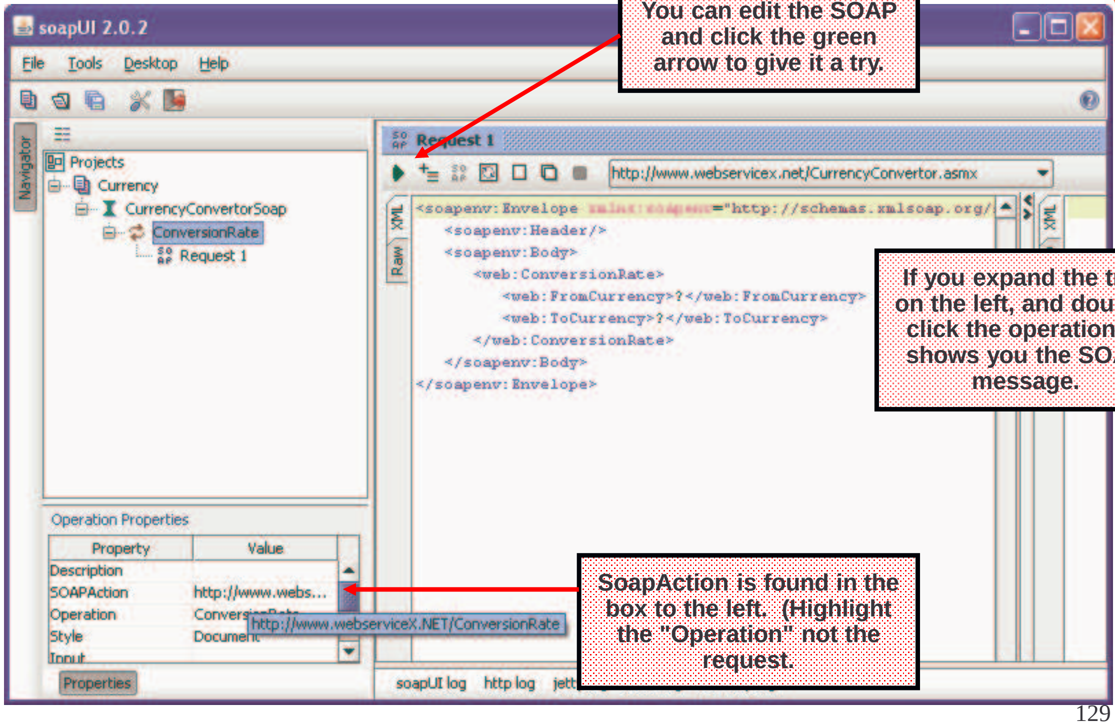
can be any name – use something you'll remember.

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)



```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
   <soapenv:Header/>
   <soapenv:Body>
      <web:ConversionRate>
         <web:FromCurrency>?</web:FromCurrency>
         <web:ToCurrency>?</web:ToCurrency>
      </web:ConversionRate>
   </soapenv:Body>
</soapenv:Envelope>
```

Operation Properties

Property	Value
Description	
SOAPAction	http://www.webs...
Operation	ConversionRate
Style	Document
Input	

http://www.webserviceX.NET/ConversionRate

Properties

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.

RPG as a Web Service Provider

Presented by

Scott Klement

http://www.scottklement.com

"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 **`/invoice`**, 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 /invoice and calls RESTFUL/INVOICE
- Our program can read the 495 (customer number) from the URL itself.

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.

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
```

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);
```

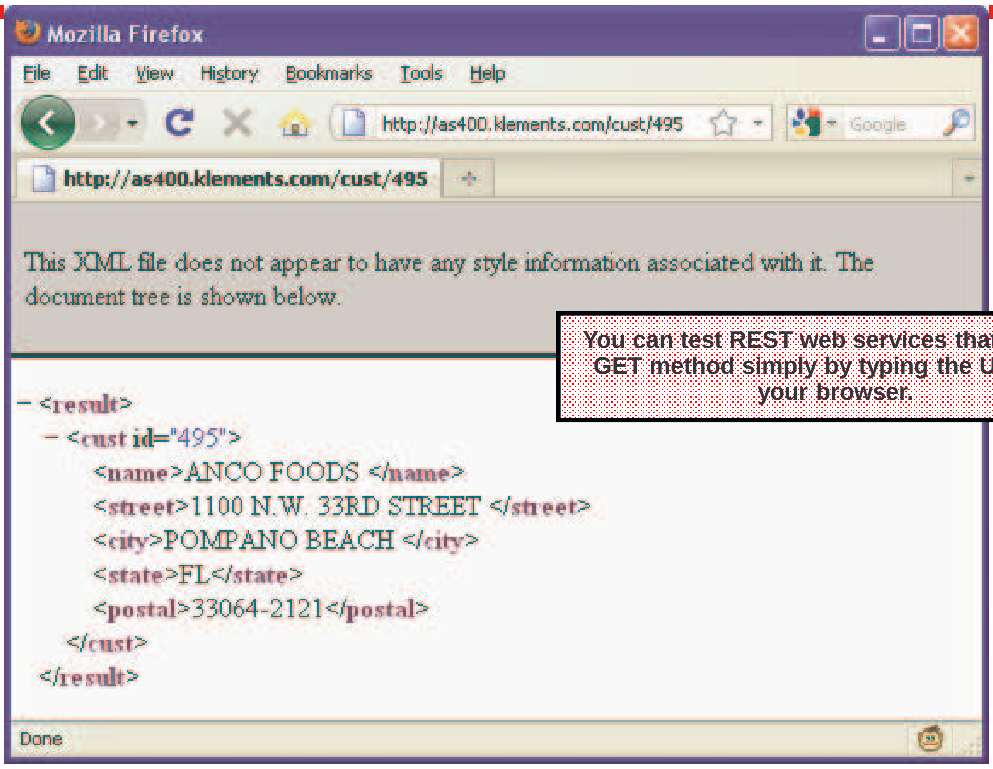
Status 200 means that all was well.

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.)

Test the REST


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://as400.klements.com/cust/495
Google
http://as400.klements.com/cust/495
Done

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <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.

Test REST with HTTPAPI

This requires HTTPAPI version 1.24 or later. This is currently in beta at: http://www.scottklement.com/httpapi/beta/

```
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')
```

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.

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

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
```

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.

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.

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

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>
```

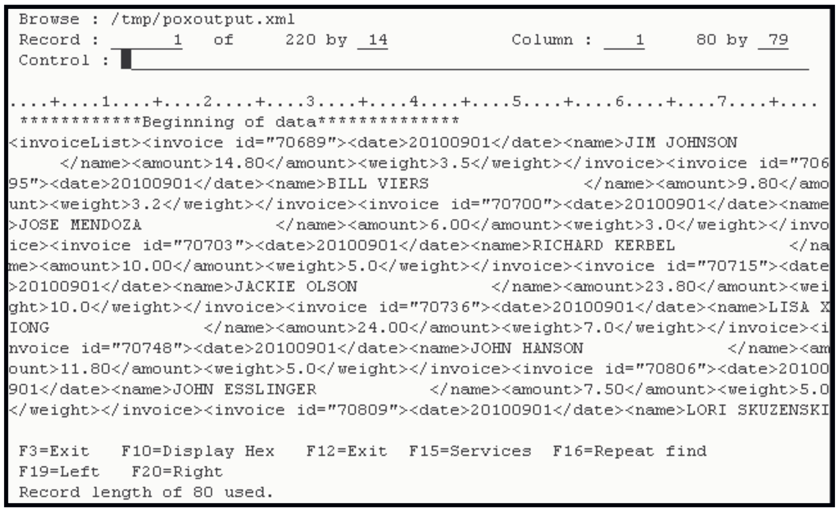
Now send it to the POX web service *(again, requires HTTPAPI 1.24+)*

```
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 it's 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.

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)
```

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                                     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.

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!

The Wizarding World

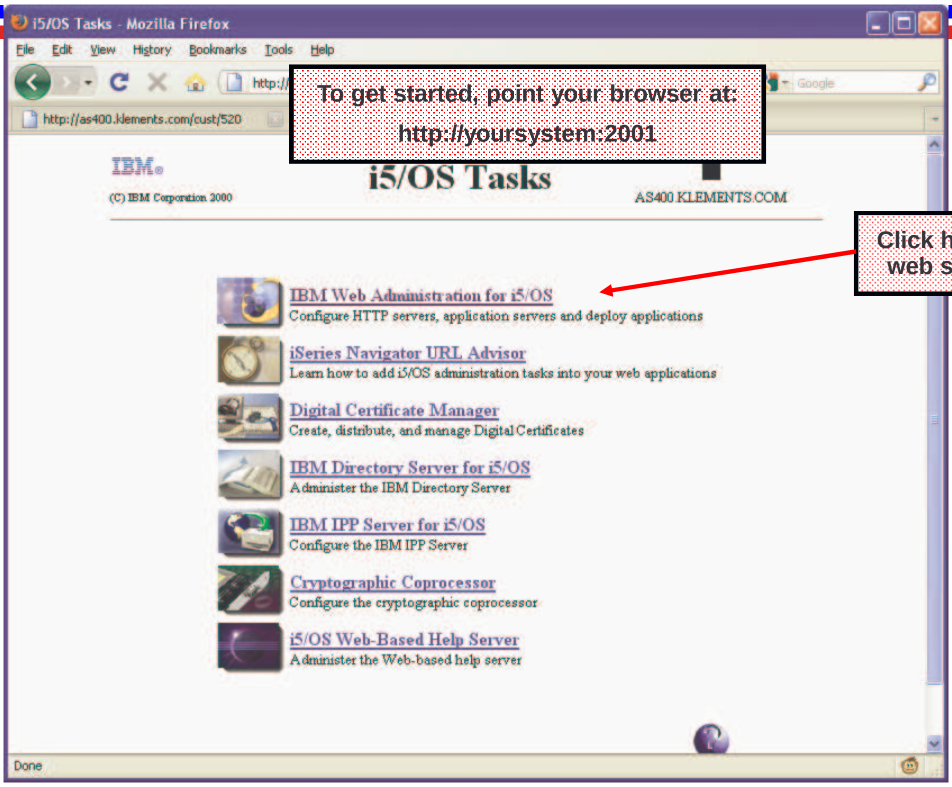
Step 1 was to make my RPG program, using parameters to return the result.

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

The IWS Wizard (1 of 13)



i5/OS Tasks - Mozilla Firefox

http://as400.klements.com/cust/520

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

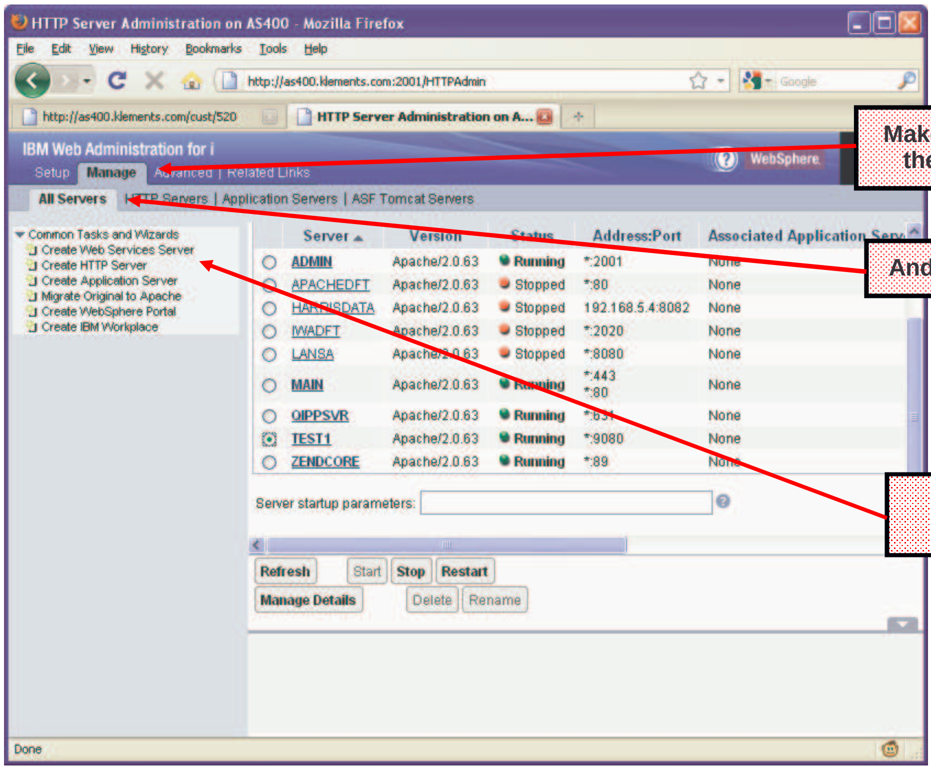
i5/OS Web-Based Help Server
Administer the Web-based help server

Done

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

Click here to get to the web services wizard.

The IWS Wizard (2 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

	Server	Version	Status	Address:Port	Associated Application Serv
○	ADMIN	Apache/2.0.63	Running	*:2001	None
○	APACHEDFT	Apache/2.0.63	Stopped	*:80	None
○	HARRISDATA	Apache/2.0.63	Stopped	192.168.5.4:8082	None
○	IWADFT	Apache/2.0.63	Stopped	*:2020	None
○	LANSA	Apache/2.0.63	Stopped	*:8080	None
○	MAIN	Apache/2.0.63	Running	*:443 *:80	None
○	QIPPSVR	Apache/2.0.63	Running	*:631	None
●	TEST1	Apache/2.0.63	Running	*:9080	None
○	ZENDCORE	Apache/2.0.63	Running	*:89	None

Server startup parameters:

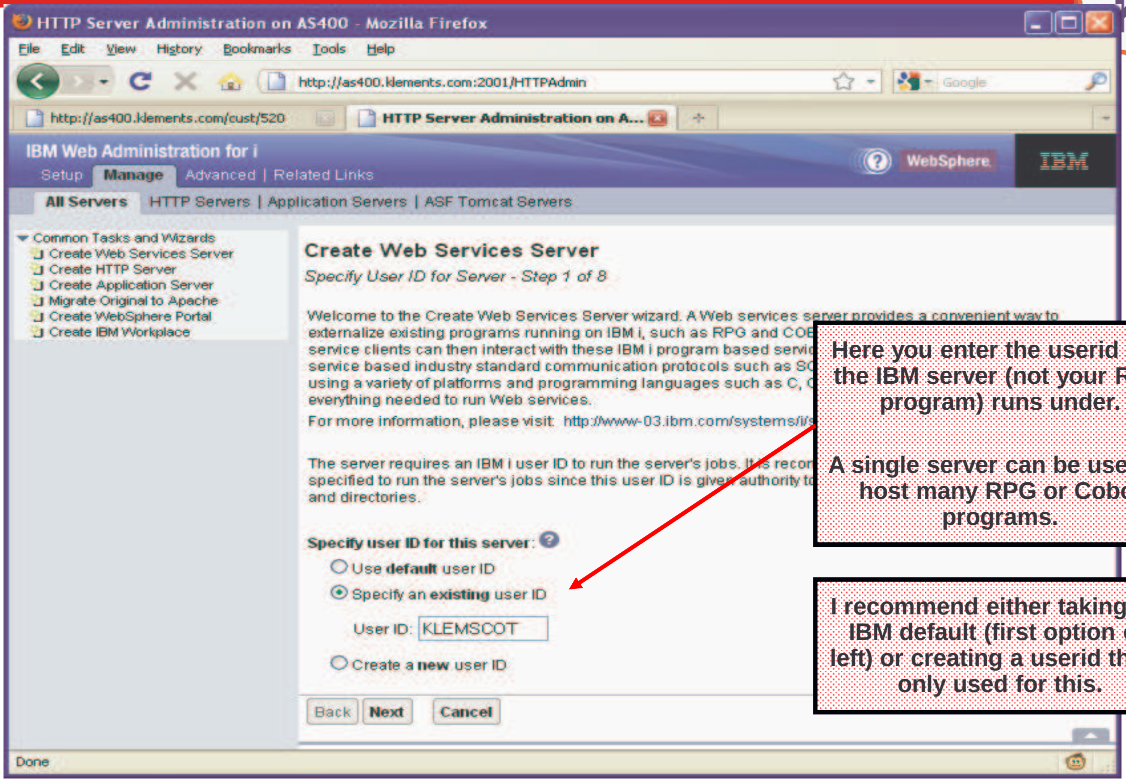
Done

Make sure you are on the "manage" tab.

And click "all servers"

Then "Create Web Services Server"

The IWS Wizard (3 of 13)


HTTP Server Administration on AS400 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://as400.klements.com:2001/HTTPAdmin
http://as400.klements.com/cust/520
HTTP Server Administration on A...
IBM Web Administration for i
Setup Manage Advanced | Related Links
WebSphere
IBM
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
Back Next Cancel
Done

Create Web Services Server

Specify User ID for Server - Step 1 of 8

Welcome to the Create Web Services Server wizard. A Web services server provides a convenient way to externalize existing programs running on IBM i, such as RPG and COB service clients can then interact with these IBM i program based servic service based industry standard communication protocols such as SC using a variety of platforms and programming languages such as C, C everything needed to run Web services.

For more information, please visit http://www-03.ibm.com/systems/i/

The server requires an IBM i user ID to run the server's jobs. It is recon specified to run the server's jobs since this user ID is given authority to and directories.

Specify user ID for this server:

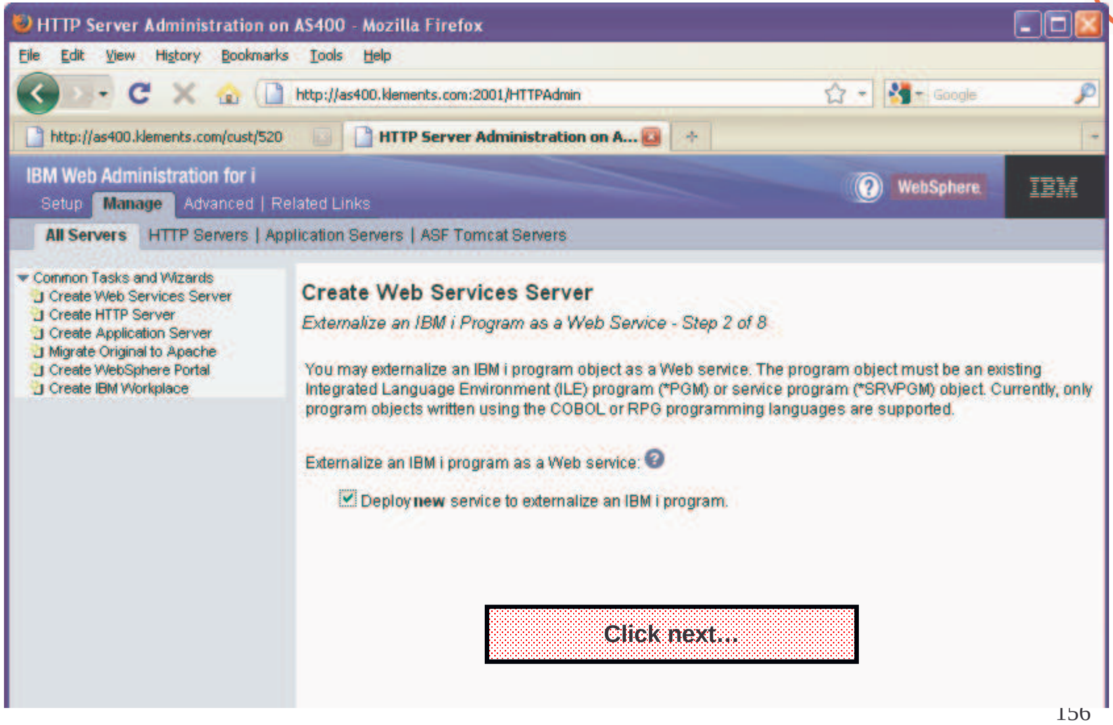
- ○ Use default user ID
- ● Specify an existing user ID
  - User ID: KLEMSCOT
- ○ Create a new user ID

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.

The IWS Wizard (4 of 13)


HTTP Server Administration on AS400 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://as400.klements.com:2001/HTTPAdmin
http://as400.klements.com/cust/520
HTTP Server Administration on A...
IBM Web Administration for i
Setup Manage Advanced | Related Links
WebSphere
IBM
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

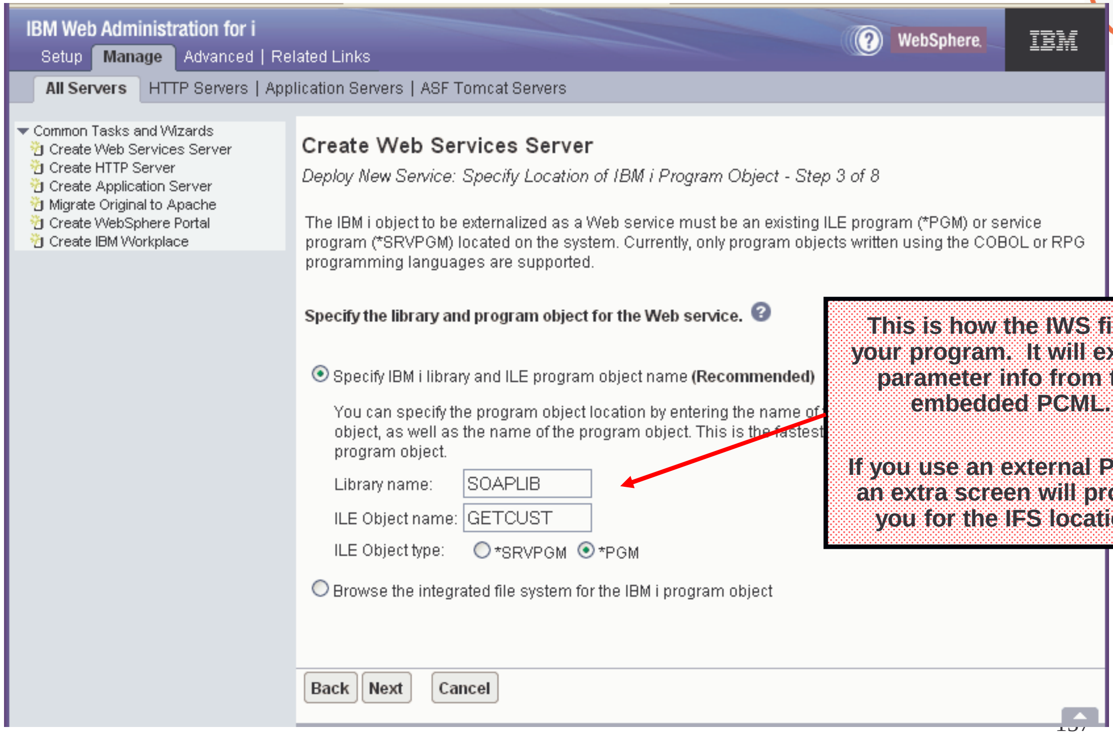
You may externalize an IBM i program object as a Web service. The program object must be an existing Integrated Language Environment (ILE) program (*PGM) or service program (*SRVPGM) object. Currently, only program objects written using the COBOL or RPG programming languages are supported.

Externalize an IBM i program as a Web service:

- ☑ Deploy new service to externalize an IBM i program.

Click next...

The IWS Wizard (5 of 13)


IBM Web Administration for i
Setup | Manage | Advanced | Related Links
WebSphere
IBM
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
Back
Next
Cancel

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 object, as well as the name of the program object. This is the fastest program object.

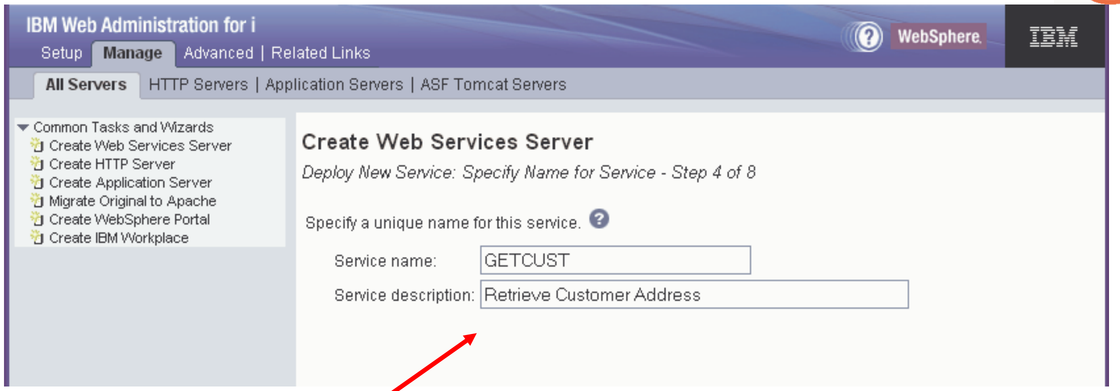
Library name:	SOAPLIB
ILE Object name:	GETCUST
ILE Object type:	*SRVPGM [✓ *PGM]

○ Browse the integrated file system for the IBM i program object

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
WebSphere
IBM
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

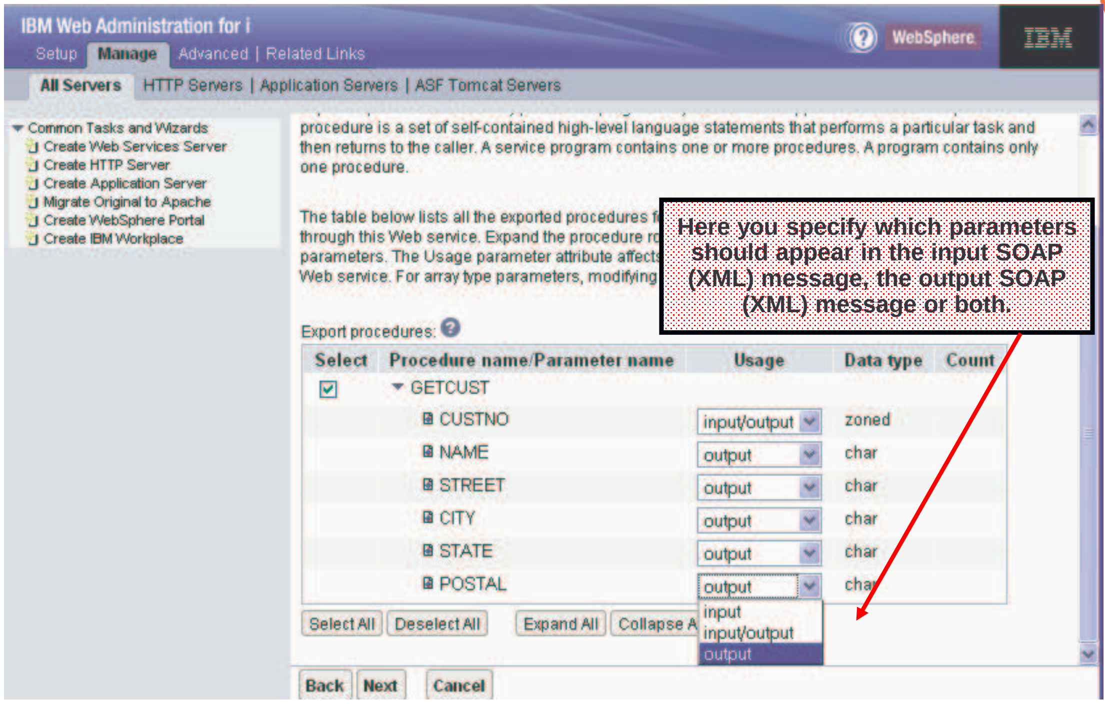
Deploy New Service: Specify Name for Service - Step 4 of 8

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 f through this Web service. Expand the procedure r parameters. The Usage parameter attribute affects Web service. For array type parameters, modifying

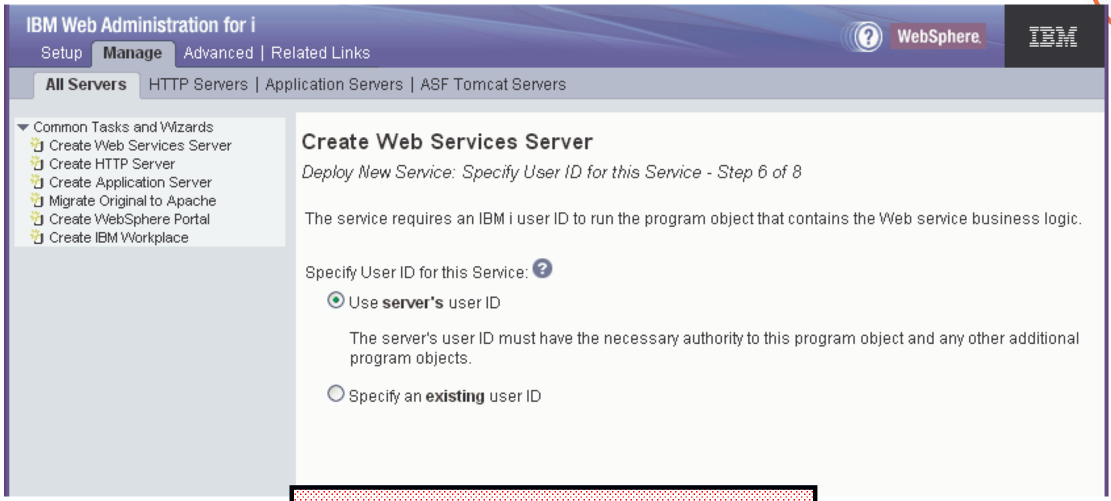
Export procedures:

Select	Procedure name/Parameter name	Usage	Data type	Count
✓	GETCUST			
	CUSTNO	input/output	zoned	
	NAME	output	char	
	STREET	output	char	
	CITY	output	char	
	STATE	output	char	
	POSTAL	output	char	

input
input/output
output

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)


IBM Web Administration for i
Setup
Manage
Advanced | Related Links
WebSphere
IBM
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 User ID for this Service - Step 6 of 8

The service requires an IBM i user ID to run the program object that contains the Web service business logic.

Specify User ID for this Service:

[✓ Use **server's** user ID]

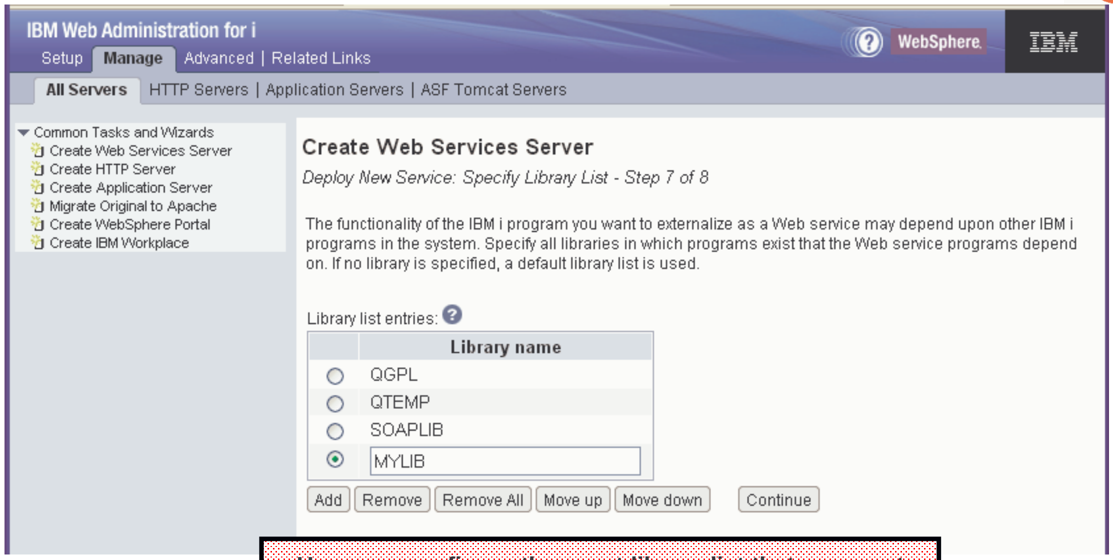
The server's user ID must have the necessary authority to this program object and any other additional program objects.

Specify an **existing** user ID

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)


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
WebSphere
IBM
Add
Remove
Remove All
Move up
Move down
Continue

Create Web Services Server

Deploy New Service: Specify Library List - Step 7 of 8

The functionality of the IBM i program you want to externalize as a Web service may depend upon other IBM i programs in the system. Specify all libraries in which programs exist that the Web service programs depend on. If no library is specified, a default library list is used.

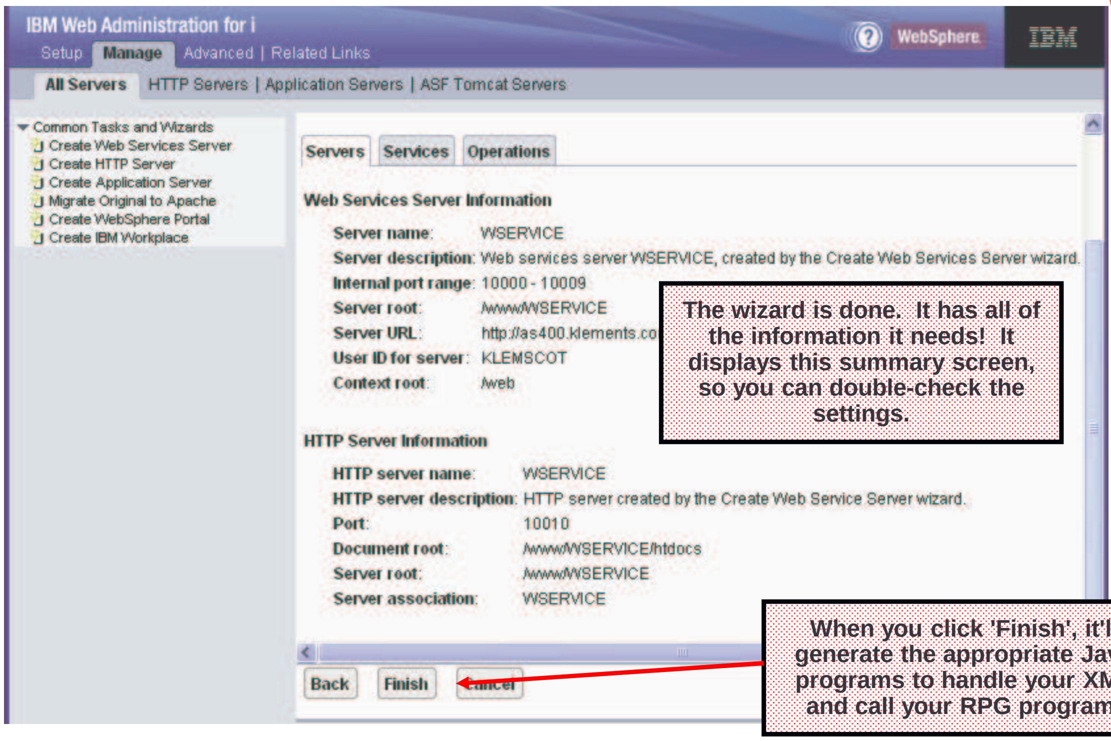
Library list entries:

	Library name
○	QGPL
○	QTEMP
○	SOAPLIB
●	MYLIB

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
WebSphere
IBM
Servers
Services
Operations
Back
Finish
Cancel

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

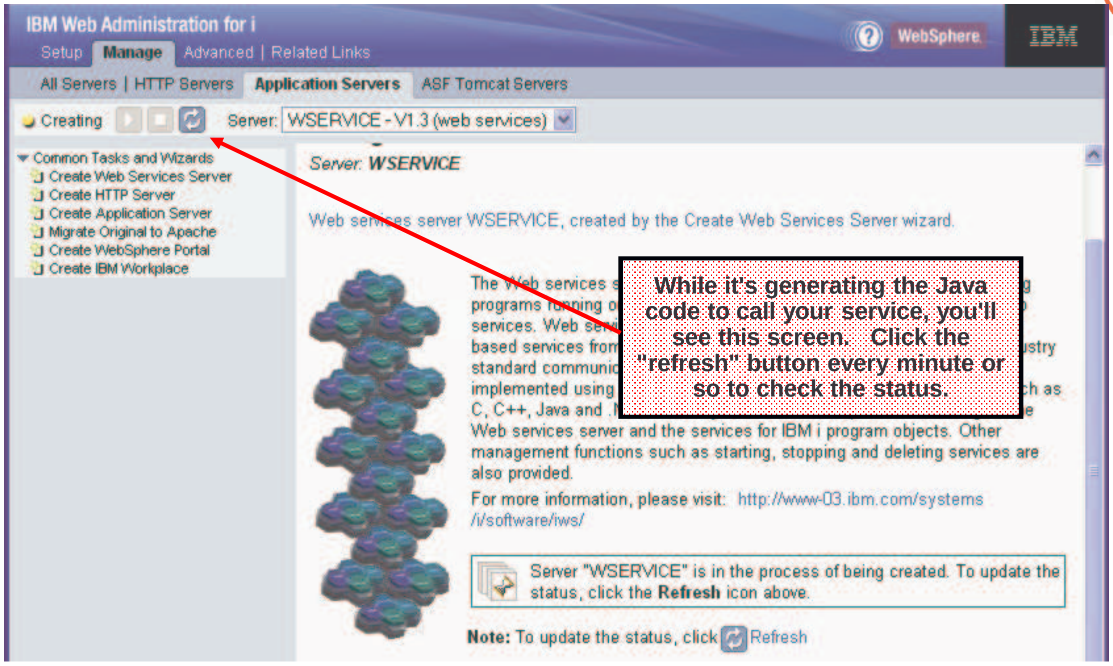
HTTP Server Information

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

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 s… programs running o… services. Web servi… based services from… standard communic… implemented using… C, C++, Java and .N… 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/

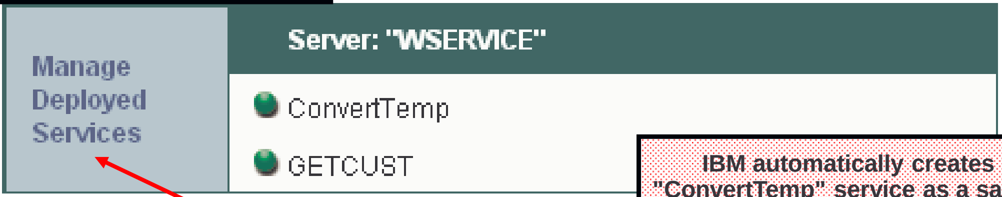
Server "WSERVICE" is in the process of being created. To update the status, click the **Refresh** icon above.

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.


Manage Deployed Services

Server: "WSERVICE"

- ConvertTemp
- GETCUST

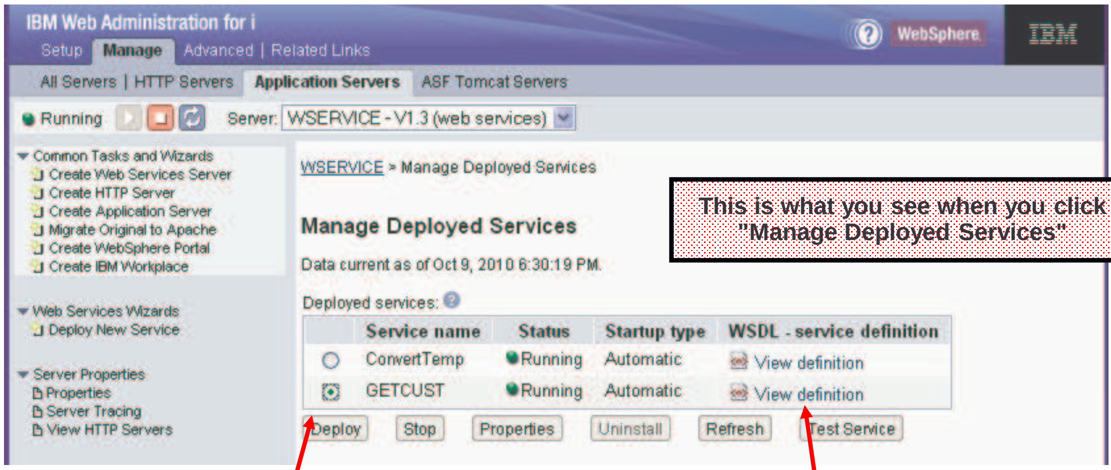
IBM automatically creates the "ConvertTemp" service as a sample to let you test your server.

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)

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

Web Services Wizards
- Deploy New Service

Server Properties
- Properties
- Server Tracing
- View HTTP Servers

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

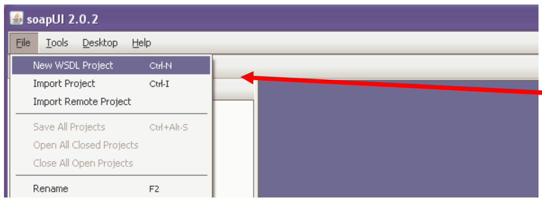
Deploy | Stop | Properties | Uninstall | Refresh | Test Service

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)



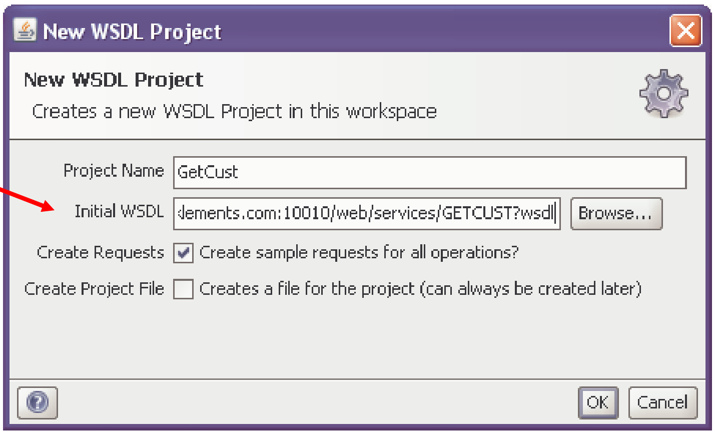
Step 1:

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.


New WSDL Project

New WSDL Project

Creates a new WSDL Project in this workspace

Project Name: GetCust

Initial WSDL: klements.com:10010/web/services/GETCUST?wsdl   Browse...

Create Requests: [✓] Create sample requests for all operations?

Create Project File: [ ] Creates a file for the project (can always be created later)

OK | Cancel

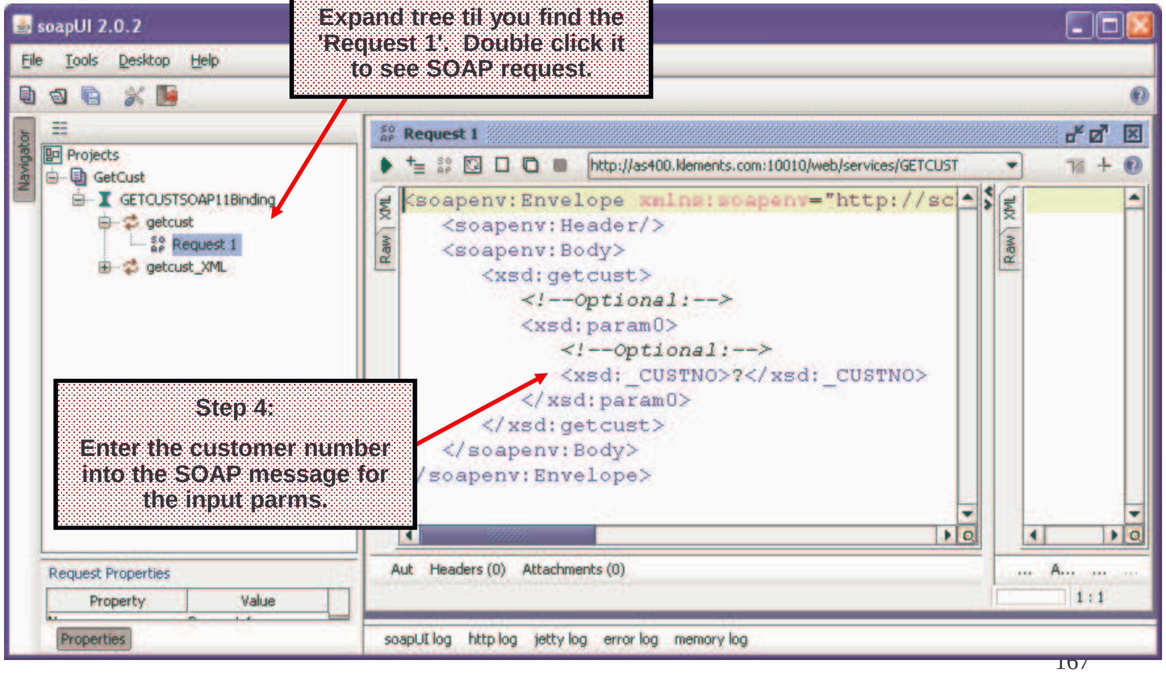
Testing SOAP with SoapUI (2 of 4)

Step 3:

Expand tree til you find the 'Request 1'. Double click it to see SOAP request.

Step 4:

Enter the customer number into the SOAP message for the input parms.

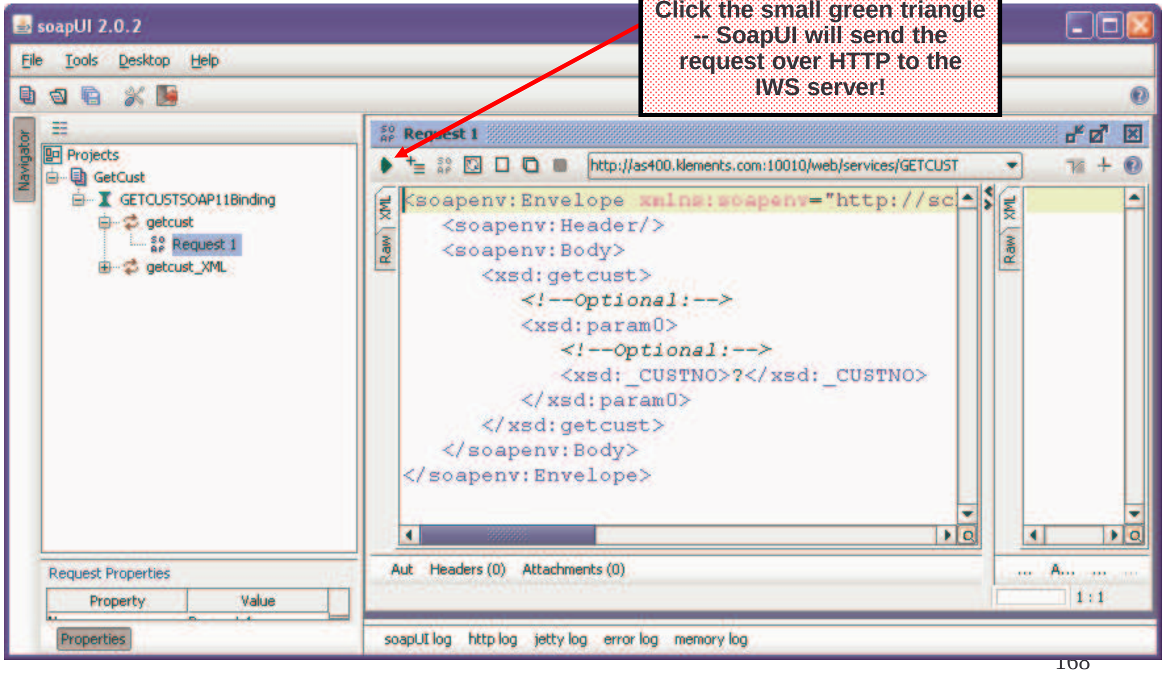


```
<soapenv:Envelope xmlns:soapenv="http://sc
   <soapenv:Header/>
   <soapenv:Body>
      <xsd:getcust>
         <!--Optional:-->
         <xsd:param0>
            <!--Optional:-->
            <xsd:_CUSTNO>?</xsd:_CUSTNO>
         </xsd:param0>
      </xsd:getcust>
   </soapenv:Body>
</soapenv:Envelope>
```

Testing SOAP with SoapUI (3 of 4)

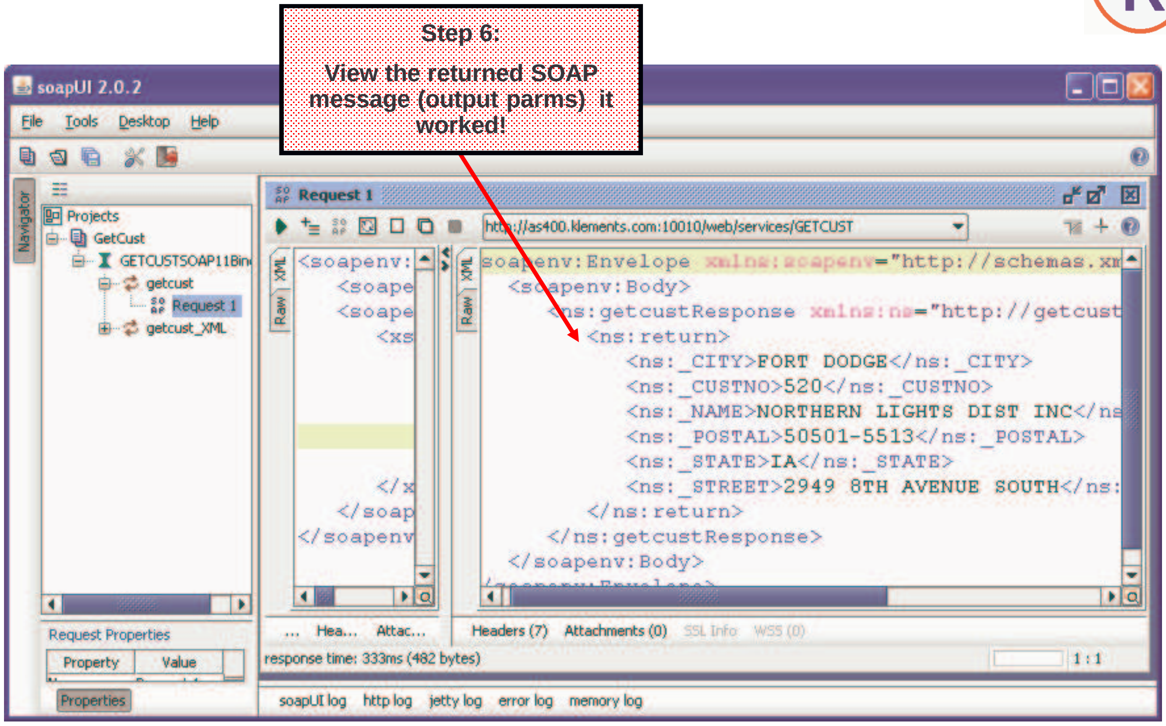
Step 5:

Click the small green triangle -- SoapUI will send the request over HTTP to the IWS server!



```
<soapenv:Envelope xmlns:soapenv="http://sc
   <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!
soapUI 2.0.2
File Tools Desktop Help
Navigator
Projects
GetCust
GETCUSTSOAP11Binding
getcust
Request 1
getcust_XML
Request Properties
Property Value
Properties
http://as400.klements.com:10010/web/services/GETCUST
Headers (7) Attachments (0) SSL Info WSS (0)
response time: 333ms (482 bytes)
soapUI log http log jetty log error log memory log

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

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 Idon't code the XML in the program?
- A: With an array!

- Q: How do 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.

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!

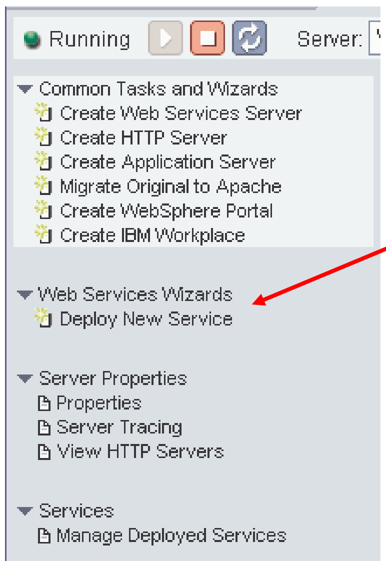
SOAPINV (invoice list) (2 of 2)

```
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.

SOAPINV In the Wizard (1 of 2)

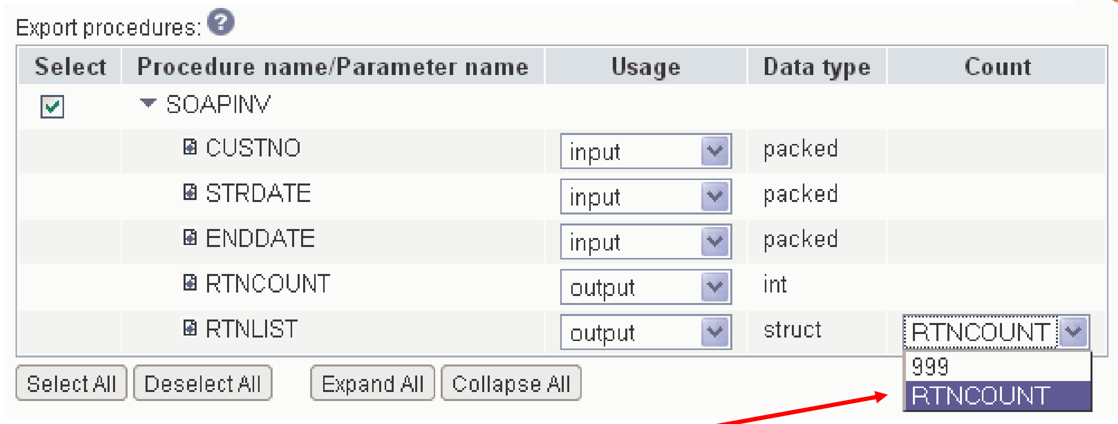


Deploy new service adds another web service to the existing server.

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.

SOAPINV In the Wizard (2 of 2)



Export procedures:

Select	Procedure name/Parameter name	Usage	Data type	Count
✓	SOAPINV			
	CUSTNO	input	packed	
	STRDATE	input	packed	
	ENDDATE	input	packed	
	RTNCOUNT	output	int	
	RTNLIST	output	struct	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.

Unfotunately, 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

"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

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
```

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.

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.

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
```

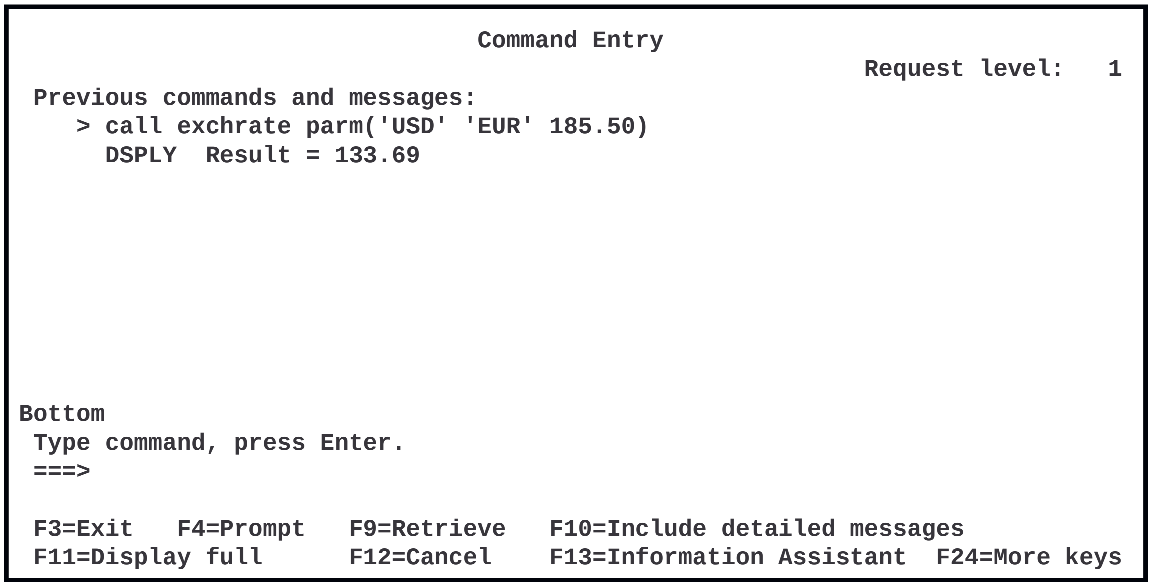
Display the error or result on the screen.

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 exchrate 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
```

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.

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)

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;
```

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 wi
           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)

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

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)*

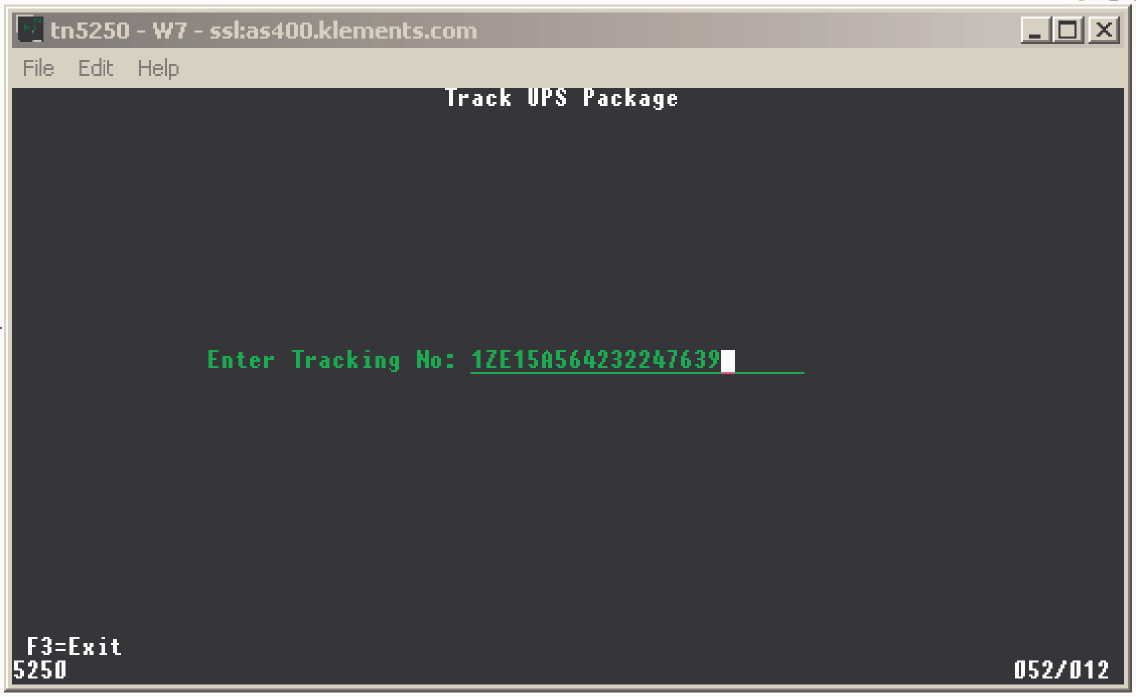
Because of (historical) import/export laws, 5722-AC3 is not shipped with OS/400. However, it's a no-charge item. You just have to order it separately from your business partner. It is included automatically in V5R4 and later as 57xx-NAE

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:

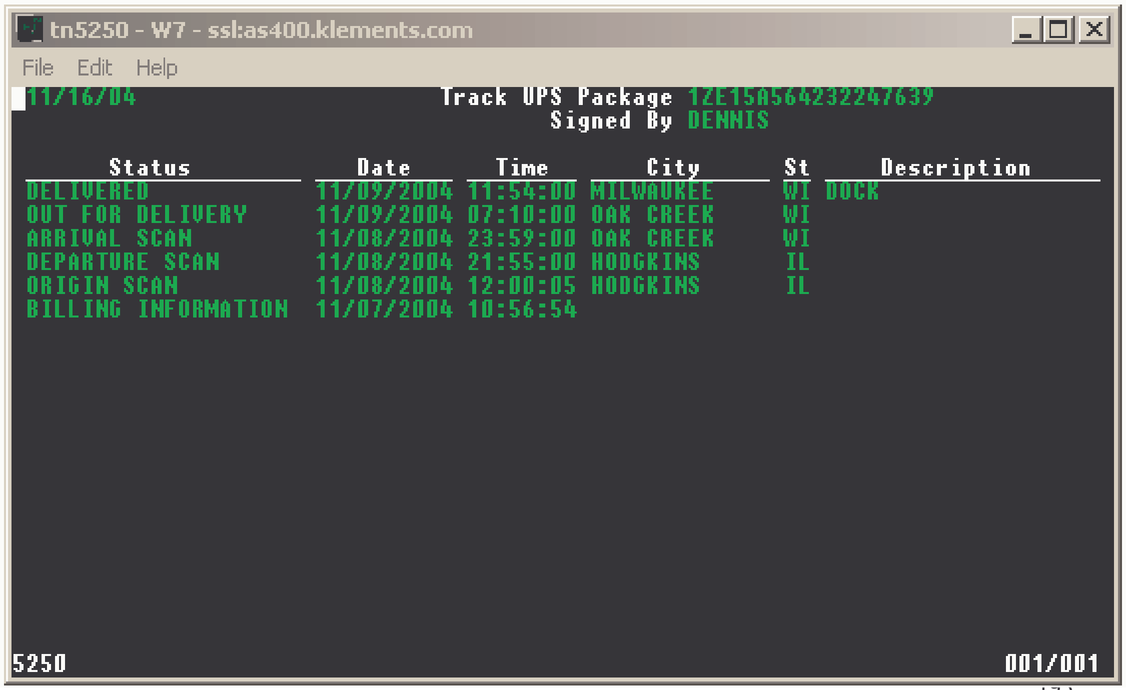
- You have to register with UPS to use their services (but it's free)
- You'll be given an access key, and you'll need to send it with each request.
- UPS requires SSL to access their web site.
- UPS does not use SOAP or WSDL for their Web services – but does use XML. Some folks call this "Plain Old 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.

UPS Example (slide 2 of 11)



```
                         Track UPS Package



          Enter Tracking No: 1ZE15A564232247639


 F3=Exit
5250                                                   052/012
```

UPS Example (slide 3 of 11)



```
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
```

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.

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.

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.

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.

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;
      endsl;

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

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

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!

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;
          endsl;

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

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

      endsl;

     /end-free
    P                  E
```

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.

HTTPAPI Information

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

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

Latest beta version:

http://www.scottklement.com/httpapi/beta

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

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

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/wsdl2rpg.html

Call a Web Service with WDSL2RPG (Thomas Raddatz)
http://systeminetwork.com/article/call-web-service-wdsl2rpg

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

Implementing PHP and RPG Web Services (Erwin Earley)
http://systeminetwork.com/article/implementing-php-and-rpg-web-services

Creating and Testing an RPG Web Service from WDSc (Jef Sutherland)
http://systeminetwork.com/article/creating-and-testing-rpg-web-service-wdsc

UPS OnLine Tools
http://www.ups.com/e_comm_access/gettools_index

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

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!