SOAP, WSDL, HTTP, XML, XSD, DTD, UDDI - what the?

By Aaron Bartell
aaron@rpg-xml.com

Copyright © Krengel Technology 2007

• Why are we at this point in technology?
• XML – Holding data the “new way”.
• XSD – Defining your xml data. ( also DTD )
• WSDL – Used at development time to describe a web service.
• SOAP – Transmission time xml.
• HTTP – How 99% of web service communication is done.
• UDDI – Web Service directory.
• Where does one start?
• PRIMARY Reason: Many platforms/languages within and external to our organizations need to exchange data and business logic easily in a platform and language independent way.

• Need better approach than previous efforts.
  – Microsoft’s DCOM (Distributed Component Object Model)
  – Java’s CORBA (Common Object Request Broker Architecture)

• Need approach that ANY language and platform can adhere to
  – A completely text based, extensible/flexible protocol (i.e. xml) is the solution.

• Web services accomplish the exact same goal as program to program communication on the iSeries. The only difference is that web services use HTTP and XML as the transport instead of OS/400.

---

**XML**

XML = Extensible Markup Language
Usage = XML holds data much like an RPG data structure

**How we store data in RPG programs**

```
  D PostAddr ds
  D residential  n
  D title  5a
  D name  15a
  D street  15a
  D cty  10a
  D state  2a
  D zip  5a
  D phone  12a dim(2)
```

What the PostAddr data structure looks like in memory.

```
........1......2......3......4......5......6......7......8......
 1Mr. Aaron Bartell  123 Center Rd  Mankato  MN   56011
          123-123-4567     567-1234
```
XML contd…

XML is composed of elements and attributes...

\[
\begin{align*}
\text{<element attribute="attribute value">element value</element>}
\end{align*}
\]

XML equivalent to RPG PostAdr data structure

\[
\begin{align*}
\text{<PostAdr residential="true">} & \\
\text{<name title="Mr."> } & \\
\text{<first> } & \\
\text{<last>Bartell</last> } & \\
\text{<street>123 Center Rk/street> } & \\
\text{<city>Mankato</city> } & \\
\text{<state>MN</state> } & \\
\text{<zip>56001</zip>} & \\
\text{<phone>112-123-1234</phone> } & \\
\text{<phone>321-321-4321</phone>} & \\
\text{</PostAdr> }
\end{align*}
\]

XML representation in stream file (i.e. /ifsfolder/PostAdr.xml)

\[
\begin{align*}
\text{<PostAdr residential="true">} & \\
\text{<name title="Mr."> } & \\
\text{<first> } & \\
\text{<last>Bartell</last> } & \\
\text{<street>123 Center Rk/street> } & \\
\text{<city>Mankato</city> } & \\
\text{<state>MN</state> } & \\
\text{<zip>56001</zip>} & \\
\text{<phone>112-123-1234</phone> } & \\
\text{<phone>321-321-4321</phone>} & \\
\text{</PostAdr> }
\end{align*}
\]

Modified or “Extended” xml, adding \texttt{<first>} and \texttt{<last>} to \texttt{<name>} element.

\[
\begin{align*}
\text{<PostAdr residential="true">} & \\
\text{<name title="Mr."> } & \\
\text{<first>Aaron</first> } & \\
\text{<last>Bartell</last> } & \\
\text{<street>123 Center Rk/street> } & \\
\text{<city>Mankato</city> } & \\
\text{<state>MN</state> } & \\
\text{<zip>56001</zip>} & \\
\text{<phone>112-123-1234</phone> } & \\
\text{<phone>321-321-4321</phone>} & \\
\text{</PostAdr> }
\end{align*}
\]

What about data types? Enter XSD’s…

XSD

XSD = XML Schema Definition

XSD’s are used to define your XML data much like DDS defines your Physical Files and what data can be stored in them. Define data types, parentage (e.g. RPG QUALIFIED DS’s), lists of valid values (e.g. AL, AK, AZ, AR, etc), patterns (e.g. only allow lower case a-z), ranges (i.e. greater than 10 but less than 100).

Can be used at both development time to know how messages should be composed, and at runtime to validate messages from other parties.

\[
\begin{align*}
\text{<schema> } & \\
\text{<element name="PostAdr">} & \\
\text{<complexType> } & \\
\text{<element name="name">} & \\
\text{<complexType> } & \\
\text{<element name="first" type="string"></element> } & \\
\text{<element name="last" type="string"></element> } & \\
\text{</complexType> } & \\
\text{</element> } & \\
\text{<element name="street" type="string"></element> } & \\
\text{<element name="city" type="string"></element> } & \\
\text{<element name="state" type="string"></element> } & \\
\text{<element name="zip" type="string"></element> } & \\
\text{<element name="phone" type="string"> minOccurs="1" maxOccurs="2"></element> } & \\
\text{</sequence> } & \\
\text{<attribute name="title" type="string" /> } & \\
\text{</complexType> } & \\
\text{</element> } & \\
\text{<element name="residential" type="boolean" /> } & \\
\text{</complexType> } & \\
\text{</schema>}
\end{align*}
\]

Why not use DTD’s?
DTD = Document Type Definition. Old technology replaced by XSD.

Biggest Downfall: Doesn’t allow the specifying of specific data types (i.e. string, integer, double, date, timestamp, boolean, etc)

Use WDSC to convert your DTD’s to XSD’s – very simple process.

```
<!DOCTYPE PostAddr [ 
  <!ELEMENT PostAddr [name,street,city,state,zip,phone+]> 
  <!ELEMENT first [#CDATA]> 
  <!ELEMENT last [#CDATA]> 
  <!ELEMENT street [#CDATA]> 
  <!ELEMENT city [#CDATA]> 
  <!ELEMENT state [#CDATA]> 
  <!ELEMENT zip [#CDATA]> 
  <!ELEMENT phone [#CDATA]> 
  <!ATTLIST PostAddr residential CDATA #REQUIRED> 
  <!ATTLIST name title CDATA #REQUIRED> 
]>
```

Elements vs. Attributes

- Rule of thumb: Use attributes at all time unless an element is warranted.
- Elements are warranted when:
  - If a piece of data needs to repeat itself multiple times within the same parent tag, it should be an element instead of an attribute.
  - If it is necessary to pass information that is not supposed to be parsed by the xml parser, the `<![CDATA[don’t “parse” <me&me>]]>` tag will need to be used. The CDATA tag is only valid in elements and cannot be used in attributes. Note that you can escape XML reserved characters that reside in attributes.
  - If there is any chance of needing future expansion or the data is complex (i.e. a postal address), attributes are not the solution for that particular piece of data and an element should be used instead. (e.g. `<phone>` could potentially need to change since something like a phone extension may need to be added)
  - Use elements if the contained data could exceed 256 characters in value. This has been devised as an acceptable maximum amount to put into attributes. There are no industry standards that dictate this opinion. To put this rule of thumb into context and relate it to another limitation in the industry, consider how some web servers limit the amount of data that can be passed on an HTTP GET command. Some don’t allow more than 1024 bytes. In the same respect, not all parsers are written by one person and there may be limitations for lengths in attributes.
Elements vs. Attributes contd...

Overboard Element Approach

```xml
<residential>true</residential>
<name>
  <title>Mr.</title>
  <first>Aaron</first>
  <last>Bartell</last>
</name>
<street>123 Center Rd</street>
<city>Mankato</city>
<state>MN</state>
<zip>56001</zip>
<phone>123-123-1234</phone>
<phone>321-321-4321</phone>
</PostAddr>
```

Overboard Attribute Approach

```xml
<PostAddr residential="true" title="Mr." firstName="Aaron" lastName="Bartell" street="123 Center Rd" city="Mankato" state="MN" zip="56001" phone1="123-123-1234" phone2="321-321-4321" />
```

Namespaces

Namespaces = Used to fully qualify elements and attributes in an xml document. Very similar to using the PREFIX keyword on a physical file’s ‘F’ spec in your RPG program.

**Fully Qualifying a Physical File’s Fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Alias</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>R DORDRPR</td>
<td>TEXT('Order Header')</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>ID</td>
<td>15F</td>
<td>COLHDG('Unique Id of row')</td>
</tr>
<tr>
<td>A</td>
<td>CRTDT</td>
<td>Z</td>
<td>COLHDG('Row Creation Date')</td>
</tr>
<tr>
<td>A</td>
<td>DUEDT</td>
<td>L</td>
<td>COLHDG('Due Date')</td>
</tr>
<tr>
<td>A</td>
<td>TOTTXT</td>
<td>9P</td>
<td>COLHDG('Total Amount')</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Alias</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>K ID</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Alias</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>R IMPROD</td>
<td>TEXT('Inventory Product')</td>
</tr>
<tr>
<td>A</td>
<td>ID</td>
<td>15F</td>
</tr>
<tr>
<td>A</td>
<td>CRTDT</td>
<td>Z</td>
</tr>
<tr>
<td>A</td>
<td>PROCD</td>
<td>5A</td>
</tr>
<tr>
<td>A</td>
<td>PROID</td>
<td>20B</td>
</tr>
<tr>
<td>A</td>
<td>QOH</td>
<td>5P</td>
</tr>
</tbody>
</table>

Fields not qualified = Compile time errors

```rpg
FORHDRPF if e k disk
FINHDRPF if e k disk
/free
display 'Program ran';
*inlr = *on;
/end-free
```

Fields qualified = Compile successful

```rpg
FORHDRPF if e k disk prefix(C)
FINHDRPF if e k disk prefix(I)
/free
display 'Program ran';
*inlr = *on;
/end-free
```
Namespaces contd...

- Attribute ‘xmlns’ is used in an element to declare a namespace (say prefix).
- Syntax is `xmlns:namespace-prefix="namespaceURI`
  - Omitting ‘namespace-prefix’ declares that this is the default namespace for the document.
  - ‘namespaceURI’ is not used to look up information about the namespace, it is simply used as a
    convention to uniquely identify an organization’s elements and attributes. Sometimes though the
    namespaceURI WILL point to an actual web page that contains the XML Schema for the declared
    namespace.
- In the below example the <phone> element needed to be prefixed or qualified with a namespace
  because the makeup of the vendors <phone> element is different from that of your companies <phone>
  element. When the parser gets to <vendor:phone ...> it will know to look for attributes areaCode,
  phoneNumber, and ext instead of accessing the elements content.

```xml
<envelope xmlns="http://mowayr4lawn.com/order.vi.3.xsd"
          xmlns:vendor="http://vendor.com/vendor_versions.7.xsd">
  <order dueDate="2006-10-19">
    <vendor:phone areaCode="612" phoneNumber="112-8212" ext="5908"/>
  </order>
</envelope>
```

WSDL

- **WSDL = Web Services Description Language**
  - Used at development time to describe a web service program, including the names of
    procedures, input/output parameters, address of web service, and what enveloping
    mechanisms and transport will be used (i.e. SOAP over HTTP).
  - WSDL’s are very similar in concept to RPG ILE Service Programs
    - Both declare the names of “procedures” that are publicly available.
      - Using command DSPSRVPGM to list exported sub procedures and an RPG ILE
        prototype would be equal to a WSDL <portType> element.
    - Both detail the input/output parameters of the procedures.
      - Using RPG data structures as parameters are the same as WSDL’s <message>
        element.
    - Both state where the program/web service are located
      - Using DSPSRVPGM to see the library of the service program is similar to WSDL’s
        <service> and <soap:address> element.
The binding tag is not as easily related to the RPG environment. Its purpose is to connect the <service> to the <portType> and define the enveloping mechanism and transport to be used (i.e. SOAP envelopes over HTTP).

Note the <binding> type attribute specifies a portType of "rxs:ORDSV" giving it logical connection to portType "ORDSV".

Note the <binding> element specifies an <operation> named "Ord_calcPrice" which is referencing the same named portType operation.

The soap:binding tag declares that HTTP will be used as the transport. The soap:operation and soap:body declare that "Document Literal" will be used for the SOAP body. The other popular option for the SOAP body is RPC/Encoded.

What encoding to use? Excellent article explaining it: http://tinyurl.com/b4uql
WDSC has some great graphical utilities for creating WSDL’s. Below is an image showing the WSDL described on the previous slides.

**SOAP**

- **SOAP = Simple Object Access Protocol**
- SOAP is simply XML that envelopes the actual business related XML you are sending across the wire (i.e. Ord_calcPrice). SOAP aims to emulate program to program communication that happens across platform and language boundaries.
- The SOAP specification is not “simple” in reality - though if stripped down correctly can be more easily used. SOAP was never meant for humans to read and make sense of, instead it was meant to be consumed by tooling.
- Opinion: SOAP for straight forward XML messages is overkill, but being that SOAP is THEE standard for web service communication it is sometimes a necessary evil. Most SOAP implementations DO NOT take advantage of the full feature set of SOAP and could instead simply use the below Simple XML Request to accomplish the same end goal.

### RPC/Encoded SOAP XML Request

```
<SOAP-ENV:envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <Ord_calcPrice xsi:type="xsd:string">233</Ord_calcPrice>
    <orderDate xsi:type="xsd:dateTime">2006-10-10</orderDate>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

### Simple XML Request

```
Ord_calcPrice>
  <orderID>123</orderID>
  <dueDate>2006-10-26</dueDate>
</Ord_calcPrice>
```

### Document/Literal SOAP XML Request

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <Ord_calcPrice xsi:type="xsd:string">233</Ord_calcPrice>
    <orderID>123</orderID>
    <dueDate>2006-10-26</dueDate>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
Parts of a SOAP message

SOAP Envelope element (i.e. `<soap:Envelope>`) is used to hold the other components of the SOAP message and declare namespaces `xmlns:soap="http://www.w3.org/2001/12/soap-envelope"` and `soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding"`

SOAP Header element (i.e. `<soap:Header>`) is optional and used to hold application specific information like authentication, but it is more common to use HTTP Basic Authentication or rely on SSL client certificates than `<soap:Header>` for that purpose. Most SOAP messages do not have a SOAP Header in them.

SOAP Body element (i.e. `<soap:Body>`) is mandatory to include on any SOAP message and is where the payload of XML id specified, or rather where you put the XML specifically related to your request (i.e. `<Ord_calPrice>`).

SOAP Fault element (i.e. `<soap:Fault>`) is optional and used to carry a response message about something gone wrong in the request. This could be related to an error getting thrown onto the OS/400 call stack and being passed to your program to potentially be displayed to the end user.

```xml
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Header>
  ...
  </soap:Header>
  <soap:Body>
    ...
  </soap:Body>
</soap:Envelope>
```

HTTP

- HTTP = HyperText Transfer Protocol
  - Used to send and receive XML from the client to the server. If you are making a request then your iSeries is the "client". If you have a web service program on your iSeries that other programmers call then your iSeries is the "server" or "end point".
  - It is becoming more common to use HTTPS (or SSL) which gives a higher level of security. SSL certificates are fully supported by the iSeries.

HTTP Request

```xml
POST /myWS/myWS HTTP/1.0
Host: 172.29.136.43:8383
User-agent: Mozilla/4.0 [compatiable; MSIE 5.5; Windows 95]
Content-type: application/x-www-form-urlencoded
Content-length: 358

<postAddr residential="true">
  <name title="Mr">
    Aaron/Bartell</name>
  <addr>
    <street>122 Center Rd</street>
    <city>Mt.40</city>
    <state>WV</state>
    <zip>26104</zip>
    <phone>312-312-3222</phone>
    <phone>312-321-4332</phone>
  </addr>
</postAddr>
```

HTTP Response

```
HTTP/1.1 200 OK
Date: Tue, 10 Oct 2006 14:22:48 GMT
Server: Apache
Content-length: 84
Connection: close
Content-type: text/html; charset=ISO-8859-1

<response status="success">PostAddr for Aaron Bartell has been processed.</response>
```
• **UDDI** = Universal Description, Discovery and Integration. Used to centrally store your WSDL documents.

• **Facts**
  - First created primarily for the public to "discover" web services that businesses may want to use, but private or internal UDDI implementations have become the norm.
  - Use is NOT widespread. Most companies are just using a handful of web services and a UDDI server implementation is not necessary or is being deemed overkill.
  - Doesn't allow for non-'web service' services to be included in the repository. Would be great to have one technology to document all programmatical processes in an organization regardless of platform.

• Sometimes used for runtime lookup of information. For example, if web service Ord_calcPrice1 is down then your program will ask the UDDI server which web service to try next (e.g. maybe Ord_calcPrice2). While this gives the calling program redundancy of remote servers to call, it has some inherent flaws, like what happens if the UDDI server is down?

• **Future Use? Here is what COULD be documented in UDDI but isn't.**
  - Internal AJAX web service documentation
  - Google web services - http://code.google.com/

Where does one start?

• Web Services is a loosely defined term and you can use what works best for you and your trading partners.
  - **Initial Approach**
    - Simply send XML via HTTP without SOAP, without defining it first via XSD, without using Namespaces, without creating a WSDL and without publishing it to a UDDI repository. Be mindful of all the technologies so you can plan for the future, but it is better to test the technologies incrementally.
  - **Next approach**
    - Determine what you want to pass for data and create an XSD (XML Schema Definition) using WDSC. (tutorial coming soon!)
    - Create what’s called an *instance* of the XML by right clicking on the XSD and choosing the Generate->XML File option. This will give you the full structure of XML to be passed allowing you to proceed and put it into your RPG program.
  - **Final Approach**
    - Using WDSC’s graphical editors, create a WSDL from the ground up and test it by right clicking Web Services->Test with Web Services Explorer. (tutorial coming soon!)
Aaron Bartell
aaron@rpg-xml.com
Senior Developer of RPG-XML Suite (www.rpg-xml.com) and
owner of www.mowyourlawn.com