

Web Services Coding Tips & Tricks

André Tost

IBM Software Group

atost@us.ibm.com





Agenda

- Introduction
- Encoding and Invocation Styles
 - “Document/literal” *versus* “rpc/encoded”
- Imports
- Collections
- JAX-RPC Handlers
- SOAP Headers
- Custom Serialization
- Binary Data
- Roundtripping issues
- Exception handling
- Array gotcha
- Resources





What Are “Web Services Tips & Tricks”?

- Collection of articles published since September 2003 on developerWorks web services domain
 - http://www-106.ibm.com/developerworks/views/webservices/articles.jsp?sort_order=desc&expand=&sort_by=Date&show_abstract=true&view_by=Search&search_by=tip%3A
- Written by developers and consultants
- Addressing real world issues and concerns that we have run into at customers and partners
- Focus is on coding/implementation, not on architecture/design
- We can only address some of the tips here, and new ones are coming out all the time, so we encourage you to go and check the latest list!



Encoding and Invocation Styles

- What is the problem?
 - Confusion about when to use which style
 - Java tools favored RPC/SOAP encoded
 - .Net tools favored Document/Literal
- How is it solved?
 - We have settled on Document/Literal as the dominant combination of encoding and invocation styles
 - Most if not all Java environments support this now
 - Interoperability has become much less of a problem
- But...
 - There are situations in which Document style does not work (or at least not well)
 - Method overloading
 - SOAP With Attachments
 - Existing web services still use SOAP encoding



WSDL Encoding Example

SOAP Encoded

```
<binding name="TemperatureConverterBinding" type="tns:TemperatureConverter">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="fahrenheitToCelsius">
    <soap:operation soapAction="" style="rpc"/>
    <input name="fahrenheitToCelsiusRequest">
      <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:TemperatureConverter" use="encoded"/>
    </input>
```

Literal

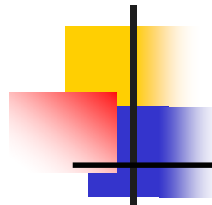
```
<binding name="CSharpTempConverterSoap" type="s0:CSharpTempConverterSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <operation name="celsiusToFahrenheit">
    <soap:operation soapAction="http://wsbootcamp.com/webservices/celsiusToFahrenheit"
      style="document" />
    <input>
      <soap:body use="literal" />
    </input>
```





Encoding in WSDL

- Each operation's *input* and *output* elements in the binding must be declared to *use* either *literal* or *encoded* style in the binding
- This is done *via* the *body* element from the SOAP binding namespace
- Encoded
 - If *use* = "encoded" then each message part refers to a base type or an abstract type (in the types section) using the *type* attribute on the *part* element (not the *soap:body* element).
 - The encoding is specified in the *encodingStyle* attribute
- Literal
 - If *use* = "literal" then each message part references a concrete schema definition using either the *element* or the *type* attribute on the *part* element (not the *soap:body* element)



Encoding

- SOAP encoding is outlawed by the Basic Profile
 - It defined a new type system, together with rules for how to encode it
 - Vendors interpreted it differently, and no language specific mappings were defined
 - This resulted in numerous interoperability problems
- Literal encoding means no encoding at all
 - Created artifacts are direct instances of the XML Schema type definitions referred to in WSDL messages
 - This means that we settle down on something that is well established and supported, without creating new type systems and encoding rules
- Hence, for the remainder of the discussion, we will assume literal encoding





Invocation Styles

- There are actually three different styles, two of them well defined, one of them not so well
 - Defined in the spec are “document” and “RPC”
 - “wrapped document” style is commonly used, but not explicitly specified or standardized
- We’ll go through examples for all three
 - WSDL definition
 - SOAP message
- We will focus on the core pieces and not cover every single detail





RPC Style

- RPC implies that the web service <operation> is representing a procedure that can be invoked remotely
 - A procedure has a name
 - A procedure takes zero or n parameters
 - A procedure returns a result
- For an RPC style service, the following rules are defined:
 - The procedure name is the operation name and is sent as the root element within the SOAP body
 - This root element contains one child element for each parameter
 - Parameters can contain additional children if they are complex types
 - The response message is directly contained in the SOAP body



RPC Style *(Continued)*

- The WS-I Basic Profile defines a few additional rules for RPC style services
- One of them is important:

“An rpc-literal binding in a DESCRIPTION MUST refer, in its soapbind:body element(s), only to wsdl:part element(s) that have been defined using the type attribute.” (R2203)

- This means that for multiple parameters, each of them is defined as a <part> within the request message
- Each of these <part> elements refers directly to a type, not to a global element defined in a schema



Java Used for Creating Examples

```
public class TheDateBean {  
    public java.util.Date getTheDate(String firstString,  
                                     String secondString) {  
        return new java.util.Date();  
    }  
}
```

- Note: This is an admittedly simple example. Using a complex example here would not have added much to the messaging style discussion. Using two parameters on the interface and using one non-trivial type is assumed to be sufficient for the sake of this discussion.





RPC Style Example – WSDL

```

<wsdl:definitions .../>
<wsdl:message name="getDateRequest">
  <wsdl:part name="arg_0_0" type="xsd:string"/>
  <wsdl:part name="arg_1_0" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="getDateResponse">
  <wsdl:part name="getDateReturn" type="xsd:date"/>
</wsdl:message>
<wsdl:portType name="TheDateBean">
  <wsdl:operation name="getDate" parameterOrder="arg_0_0 arg_1_0">
    <wsdl:input name="getDateRequest" message="intf:getDateRequest"/>
    <wsdl:output name="getDateResponse" message="intf:getDateResponse"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="TheDateBeanSoapBinding" type="intf:TheDateBean">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getDate">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getDateRequest">
      <wsdlsoap:body use="literal" namespace="http://thepack"/>
    </wsdl:input>
    <wsdl:output name="getDateResponse">
      <wsdlsoap:body use="literal" namespace="http://thepack"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>

```



RPC Style Example – SOAP

- Request envelope

```
<soapenv:Envelope
  xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
  xmlns:soapenc=http://schemas.xmlsoap.org/soap/encoding/
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <getTheDate xmlns="http://thepack">
      <arg_0_0 xmlns="">joe</arg_0_0>
      <arg_1_0 xmlns="">bill</arg_1_0>
    </getTheDate>
  </soapenv:Body>
</soapenv:Envelope>
```

- Note that the SOAP message contains elements that are not defined in the <types> section of the WSDL file



Makes validation the message harder for the SOAP engine



Document Style

- The XML that is sent within the SOAP body is an instance of the defined WSDL message
 - Allows to have more than one child element to body
 - RPC allows only one child, *i.e.* the element named after the operation
- Interpretation of the message content completely left to the receiver
 - Does not imply that there are parameters, procedure names, *etc.*
- Generally harder to manage by the service provider
 - Must parse entire message to identify which service implementation to invoke
 - Danger of ambiguous messages
 - Example: `deleteOrder(Order)` and `addOrder(Order)` may result in the same SOAP message at runtime

Document Style Example – WSDL (not BP compliant!!)

```

<wsdl:definitions ...">
<wsdl:message name="getDateResponse">
  <wsdl:part name="getDateReturn" type="xsd:date"/>
</wsdl:message>
<wsdl:message name="getDateRequest">
  <wsdl:part name="arg_0_0" type="xsd:string"/>
  <wsdl:part name="arg_1_0" type="xsd:string"/>
</wsdl:message>
<wsdl:portType name="TheDateBean">
  <wsdl:operation name="getDate" parameterOrder="arg_0_0 arg_1_0">
    <wsdl:input name="getDateRequest" message="intf:getDateRequest"/>
    <wsdl:output name="getDateResponse" message="intf:getDateResponse"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="TheDateBeanSoapBinding" type="intf:TheDateBean">
  <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getDate">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getDateRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getDateResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>

```



Document Style Example – SOAP

- SOAP request envelope

```
<soapenv:Envelope
  xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
  xmlns:soapenc=http://schemas.xmlsoap.org/soap/encoding/
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <arg_0_0 xmlns="">joe</arg_0_0>
    <arg_1_0 xmlns="">bill</arg_1_0>
  </soapenv:Body>
</soapenv:Envelope>
```

- Note that there is no indicator for what operation is invoked!





Document Style – Basic Profile

- The Basic Profile adds some rules for document style services
- Here are the two most important ones:

“A document-literal binding in a DESCRIPTION MUST refer, in each of its soapbind:body element(s), only to wsdl:part element(s) that have been defined using the element attribute.” (R2204)

“A document-literal binding in a DESCRIPTION MUST, in each of its soapbind:body element(s), have at most one part listed in the parts attribute, if the parts attribute is specified.” (R2201)

- You have to define a wrapping element for each message part
- You can only have one part per message
- A message is represented by exactly one wrapping element
- the SOAP body has only one child element



Document Style Example – WSDL (BP compliant)

```

<wsdl:definitions ...">
  <element name="StringRequestElement">
    <complexType>
      <sequence>
        <element name="arg_0_0" nillable="true" type="xsd:string"/>
        <element name="arg_1_0" nillable="true" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="DateResponseElement">
    <complexType>
      <sequence>
        <element name="getDateReturn" nillable="true" type="xsd:date"/>
      </sequence>
    </complexType>
  </element>
  <wsdl:message name="getDateResponse">
    <wsdl:part element="DateResponseElement" name="response"/> <!-- NO type is used -->
  </wsdl:message>
  <wsdl:message name="getDateRequest">
    <wsdl:part element="StringRequestElement" name="request"/> <!-- NO type is used -->
  </wsdl:message>
  <wsdl:portType name="TheDateBean">
    <wsdl:operation name="getDate" parameterOrder="request">
      <wsdl:input name="getDateRequest" message="intf:getDateRequest"/>
      <wsdl:output name="getDateResponse" message="intf:getDateResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding ...> <!-- Binding stays the same as before -->

```



Document Style Example – SOAP (for BP Compliant)

- SOAP request envelope

```
<soapenv:Envelope
  xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
  xmlns:soapenc=http://schemas.xmlsoap.org/soap/encoding/
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <StringRequestElement>
      <arg_0_0 xmlns="">joe</arg_0_0>
      <arg_1_0 xmlns="">bill</arg_1_0>
    </StringRequestElement>
  </soapenv:Body>
</soapenv:Envelope>
```

- Note that there is STILL no indicator for what operation is invoked!



Document Style – “Wrapped”

- As mentioned before, there is no formal definition for what this is
- However, it is referenced frequently and was first established by Microsoft tooling
- WSAD 5.1 will create “wrapped” service definitions for all document style services
- WAS command line tooling allows both non-wrapped and wrapped
 - Non-wrapped is what we showed earlier
 - Non-wrapped is not basic profile compliant



•So what is it??



Document style – “Wrapped”

(Continued)

- The request message for an operation has only one part
 - as also required by the Basic Profile
- This part refers to an element, not a type
 - as also required by the Basic Profile
- The name of the wrapping element is equal to the name of the operation
- Interestingly, this leads to runtime SOAP messages that look EXACTLY like in the RPC case!!
- In summary, it is fair to say that wrapped document style includes the basic profile rules for document style, plus a naming convention for the wrapper element of the request message



Document Style “Wrapped” Example – WSDL

```
<wsdl:definitions ...">
  <wsdl:types>
    <schema ...">
      <element name="getTheDate">
        <complexType>
          <sequence>
            <element name="arg_0_0" nillable="true" type="xsd:string"/>
            <element name="arg_1_0" nillable="true" type="xsd:string"/>
          </sequence>
        </complexType>
      </element>
      <element name="getTheDateResponse">
        <complexType>
          <sequence>
            <element name="getTheDateReturn" nillable="true" type="xsd:date"/>
          </sequence>
        </complexType>
      </element>
    </schema>
  </wsdl:types>
  <wsdl:message name="getTheDateResponse">
    <wsdl:part name="parameters" element="intf:getTheDateResponse"/>
  </wsdl:message>
  <wsdl:message name="getTheDateRequest">
    <wsdl:part name="parameters" element="intf:getTheDate"/>
  </wsdl:message>
  <wsdl:portType name="TheDateBean">
    <wsdl:operation name="getTheDate">
      <wsdl:input name="getTheDateRequest" message="intf:getTheDateRequest"/>
      <wsdl:output name="getTheDateResponse" message="intf:getTheDateResponse"/>
    </wsdl:operation>
  </wsdl:portType>
```



Document Style “Wrapped” Example – SOAP

- SOAP request envelope

```
<soapenv:Envelope
  xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
  xmlns:soapenc=http://schemas.xmlsoap.org/soap/encoding/
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <getDate xmlns="http://thepack">
      <arg_0_0>joe</arg_0_0>
      <arg_1_0>bill</arg_1_0>
    </getDate>
  </soapenv:Body>
</soapenv:Envelope>
```

- You cannot distinguish this from an RPC style service





Imports

- What is the problem?
 - WSDL and XSD both define an <import> element
 - Even though they support a similar function, they are not the same
- How is it solved?
 - Well, let's have a look at some examples that show the difference





XSD Import

- Imports a namespace into a schema
 - *i.e.*, it makes the namespace known
- It does not import any file content (it is not an 'include')
- Imported namespace can be local or in a separate file
 - Optional "schemaLocation" attribute points to location
- Imported namespace must match "targetNamespace" of imported schema
- Most XML Parsers do not require importing common namespaces
 - For example, the xsd namespace itself:
`<xsd:import namespace=http://www.w3.org/2001/XMLSchema/>`





XSD Import Example

```

<?xml version="1.0" ?>
<wsdl:definitions targetNamespace="urn:listing2"
  xmlns:tns="urn:listing2"
  xmlns:listing3="urn:listing3"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <wsdl:types>
    <xsd:schema targetNamespace="urn:listing2"
      xmlns:listing3="urn:listing3"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import namespace="urn:listing3" schemaLocation="listing3.xsd"/>
      <xsd:import namespace="http://www.w3.org/2001/XMLSchema"/>
      <xsd:complexType name="Address">
        <xsd:sequence>
          <xsd:element name="streetNum" type="xsd:int"/>
          <xsd:element name="streetName" type="xsd:string"/>
          <xsd:element name="city" type="xsd:string"/>
          <xsd:element name="state" type="xsd:string"/>
          <xsd:element name="phone" type="listing3:Phone"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </wsdl:types>
  ...
</wsdl:definitions>

```



XSD Import Example *(Continued)*

```
<?xml version="1.0" ?>
<xsd:schema targetNamespace="urn:listing3"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:import namespace="http://www.w3.org/2001/XMLSchema"/>
  <xsd:complexType name="Phone">
    <xsd:sequence>
      <xsd:element name="areaCode" type="xsd:int"/>
      <xsd:element name="exchange" type="xsd:int"/>
      <xsd:element name="number" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```





WSDL Import

- Imports a piece of WSDL into a WSDL file
 - Allows referencing a schema type in the WSDL that is defined in another WSDL file
 - Will not make the content of that schema know to the locally defined schema!!
 - WSDL's <import> imports WSDL, XSD's <import> imports a schema
 - A schema cannot step outside of the schema to resolve type references
 - For resolving types, XSD's <import> is required and hence will be more common than WSDL's <import>





WSDL Import Example

```

<?xml version="1.0" ?>
<wsdl:definitions targetNamespace="urn:listing4"
  xmlns:tns="urn:listing4"
  xmlns:listing5="urn:listing5"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <wsdl:import namespace="urn:listing5" location="listing5.wsdl"/>
  <wsdl:types>
    <xsd:schema targetNamespace="urn:listing4"
      xmlns:listing5="urn:listing5"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import namespace="http://www.w3.org/2001/XMLSchema"/>
      <xsd:complexType name="Address">
        <xsd:sequence>
          <xsd:element name="streetNum" type="xsd:int"/>
          <xsd:element name="streetName" type="xsd:string"/>
          <xsd:element name="city" type="xsd:string"/>
          <xsd:element name="state" type="xsd:string"/>
          <xsd:element name="phone" type="listing5:Phone"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </wsdl:types>
  ...
  <wsdl:message name="GetPhoneResponse">
    <wsdl:part name="phone" type="listing5:Phone"/>
  </wsdl:message>
  ...
</wsdl:definitions>

```



WSDL Import Example *(Continued)*

```
<?xml version="1.0" ?>
<wsdl:definitions targetNamespace="urn:listing5"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types>
    <xsd:schema targetNamespace="urn:listing5"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import namespace="http://www.w3.org/2001/XMLSchema"/>
      <xsd:complexType name="Phone">
        <xsd:sequence>
          <xsd:element name="areaCode" type="xsd:int"/>
          <xsd:element name="exchange" type="xsd:int"/>
          <xsd:element name="number" type="xsd:int"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </wsdl:types>
</wsdl:definitions>
```





Collections

- What is the problem?
 - Java, like any good programming language, defines classes and interfaces to deal with collections of objects
 - An extended library of them can be found in the java.util package
 - Some of the most commonly used are Vector, Hashtable, Map and List
 - They are language specific and cannot easily be mapped into XML
 - JAX-RPC tooling will try to generate XML Schemas for these types, and/or use language specific mappings
 - Most likely, this will create problems when interacting with .Net
- How is it solved?
 - Use arrays!
 - Create wrapper interfaces when exposing existing code that uses collections





Collections *(Continued)*

```
import java.util.LinkedList;

public class TheCustomerService {
    public LinkedList getCustomers(String queryString) {
        Customer customer1, customer2;
        /* ... retrieve customers for query ... */
        LinkedList list = new LinkedList();
        /* iterate over the result set assigning to the list */
        list.add(customer1);
        list.add(customer2);
        return list;
    }
}
```





Collections *(Continued)*

```

<schema elementFormDefault="qualified"
  targetNamespace="http://pack"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:apachsoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://pack" xmlns:intf="http://pack"
  xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/">
  <element name="getCustomers">
    <complexType>
      <sequence>
        <element name="queryString"
          nillable="true"
          type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <complexType name="ArrayOfXSDAnyType">
    <sequence>
      <element maxOccurs="unbounded"
        minOccurs="0"
        name="item"
        nillable="true"
        type="xsd:anyType"/>
    </sequence>
  </complexType>
  <element name="getCustomersResponse">
    <complexType>
      <sequence>
        <element name="getCustomersReturn"
          nillable="true"
          type="impl:ArrayOfXSDAnyType"/>
      </sequence>
    </complexType>
  </element>
</schema>

```



Collections *(Continued)*

```
public class TheCustomerService {  
    public Customer[] getCustomers(String queryString) {  
        Customer customer1, customer2;  
        /* ... retrieve customers for query ... */  
        /* create an array large enough to hold the result set */  
        Customer[] customers = new Customer[2];  
        /* iterate over the result set assigning to the array */  
        customers[0] = customer1;  
        customers[1] = customer2;  
        return customers;  
    }  
}
```





Collections *(Continued)*

```

<schema elementFormDefault="qualified"
  targetNamespace=http://pack
  xmlns=http://www.w3.org/2001/XMLSchema
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://pack" xmlns:intf="http://pack"
  xmlns:wSDL="http://schemas.xmlsoap.org/wsdl/">
  <element name="getCustomers">
    <complexType>
      <sequence>
        <element name="queryString"
          nillable="true"
          type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <complexType name="Customer">
    <sequence>
      <element name="customerID"
        nillable="true"
        type="xsd:string"/>
    </sequence>
  </complexType>
  <element name="getCustomersResponse">
    <complexType>
      <sequence>
        <element maxOccurs="unbounded"
          name="getCustomersReturn"
          type="impl:Customer"/>
      </sequence>
    </complexType>
  </element>
</schema>

```



Collections *(Continued)*

```
public class TheCustomerServiceWrapper {
    protected TheCustomerServiceLinkedList innerService = new
        TheCustomerServiceLinkedList();

    public Customer[] getCustomers(String queryString) {
        return (Customer[]) innerService.getCustomers(queryString).toArray(new
            Customer[0]);
    }
}
```





JAX-RPC Handlers

- What is the problem?
 - Need to 'manage' request and response messages
 - Reading what's already in there, for example
 - ✓ Log each time a web service is used
 - ✓ Monitor health of service and response time
 - Adding information that is not yet there, for example
 - ✓ Creating and processing header data introduced by new web services specifications, *e.g. WS-Security, WS-ReliableMessaging, etc.*
 - ✓ Create a layer that deals with contextual information which is not part of the business interface
- How is it solved?
 - JAX-RPC defines a handler interface that allows interception of outgoing and incoming messages
 - Can be configured in web services deployment descriptors
 - No impact on service consumer or provider implementation
 - Provides access to message context
- But...
 - Cannot change structure of the message
 - No defined way for client to communicate with handler
 - Server side can use `javax.xml.rpc.server.ServiceLifecycle`



JAX-RPC Handlers *(Continued)*

```
public class PerformanceHandler extends GenericHandler {  
  
    public boolean handleRequest(MessageContext context) {  
        try {  
            Date startTime = new Date();  
            context.setProperty("startTime", startTime);  
        } catch (Exception x) {  
            // insert error handling here  
            x.printStackTrace();  
        }  
        return true;  
    }  
  
    public boolean handleResponse(MessageContext context) {  
        try {  
            Date startTime = (Date)context.getProperty("startTime");  
            Date endTime = new Date();  
            long elapsedTime = endTime.getTime()-startTime.getTime();  
            logger.write("Elapsed time is "+elapsedTime+"\n");  
        } catch (Exception x) {  
            // insert error handling here  
            x.printStackTrace();  
        }  
        return true;  
    }  
}
```





SOAP Headers

- What is the problem?
 - Need to deal with 'out-of-band' data
 - Contextual information, *e.g.* 'cookies'
 - QoS style data, for example
 - ✓ Security
 - ✓ Routing
 - ✓ Message handshake
 - ✓ Transactions
 - ✓ ...
 - Need API to insert / process this information from within JAX-RPC
- How is it solved?
 - JAX-RPC handlers
 - Use SAAJ to access SOAP message





SOAP Headers *(Continued)*

```
public boolean handleRequest(MessageContext arg) {
    if (arg instanceof SOAPMessageContext) {
        SOAPMessageContext context = (SOAPMessageContext)arg;
        try {
            SOAPHeader header = context.getMessage().getSOAPPart().getEnvelope().getHeader();
            Iterator headers =
                header.examineHeaderElements("http://schemas.xmlsoap.org/soap/actor/next");
            while (headers.hasNext()) {
                SOAPHeaderElement he = (SOAPHeaderElement)headers.next();
                System.out.println("header element name is "+
                    he.getElementName().getQualifiedName());
                System.out.println("header element value is "+he.getValue());
                // process the retrieved header element here
            }
        } catch (SOAPException x) {
            // insert error handling here
            x.printStackTrace();
        }
    }
    return true;
}
```





Custom Serialization

- What is the problem?
 - JAX-RPC is very Java centric
 - Mapping rules for WSDL-to-Java and Java-to-WSDL
 - ✓ Message parts are mapped to Java-typed parameters and return values
 - ✓ XML complex types mapped to Java classes and *vice versa*
 - No support for generic parameters
 - Some XML Schema artifacts are not supported / optional
 - No support for directly inserting XML into SOAP messages
- How is it solved?
 - `<xsd:any/>` in XML Schema is mapped to `javax.xml.soap.SOAPElement` in Service Endpoint Interface
 - Allows constructing SOAP message *via* SAAJ APIs
 - WebSphere offers a `-noDataMapping` option on `WSDL2Java`
- But...
 - Until SAAJ 1.2, there is no support to directly insert XML into `SOAPElement`
 - Can use proprietary `com.ibm.ws.webservices.xmlsoap.SOAPFactory`



Custom Serialization

XML Schema:

```
<complexType name="Order">
  <sequence>
    <element name="createDate" nillable="true" type="xsd:dateTime"/>
    <element name="customer" nillable="true" type="xsd:string"/>
    <xsd:any maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

Java:

```
public class Order implements java.io.Serializable {
  private java.util.Calendar createDate;
  private java.lang.String customer;
  private javax.xml.soap.SOAPElement[] _any;
```



Custom Serialization

```
...
// use the factory to create a new element
// the Name is created via the SOAPEnvelope, and we must define a
// namespace for each element in the body (to be WS-I compliant)
SOAPElement getOrderReturn = factory.createElement(envelope.createName(
    "getOrderReturn", "", "http://any.webservices.ibm.com"));

// now start adding children and build the full response message
// hardcode the text content for the sakes of simplicity
SOAPElement createDate = getOrderReturn.addChildElement(envelope.createName(
    "createDate", "", "http://any.webservices.ibm.com"));
createDate.addTextNode("2004-07-03T22:57:45.359Z");
SOAPElement customer = getOrderReturn.addChildElement(envelope.createName(
    "customer", "", "http://any.webservices.ibm.com"));
customer.addTextNode("Bill Smith");
SOAPElement lineItems = getOrderReturn.addChildElement(envelope.createName(
    "lineItems", "", "http://any.webservices.ibm.com"));
SOAPElement itemDesc = lineItems.addChildElement(envelope.createName(
    "itemDesc", "", "http://any.webservices.ibm.com"));
itemDesc.addTextNode("This is a line item");
SOAPElement itemNumber = lineItems.addChildElement(envelope.createName(
    "itemNumber", "", "http://any.webservices.ibm.com"));
itemNumber.addTextNode("12345");
...
```





Binary Data

- What is the problem?
 - Want to transport binary data to and from a web service
 - SOAP with Attachments defines how to do this
 - Not supported by Microsoft
 - New WS-I Profile defines rules, but also not supported by Microsoft
- How is it solved?
 - Use SOAP with Attachments if you only deal with SOAP engines that support that spec
 - JAX-RPC defines mapping rules between several MIME types and Java types
 - Transport binary data in SOAP body as `<xsd:hexBinary>`
 - This is less efficient
 - Mapped to `byte[]` in Java
 - Wait for further clarification and standardization in this space





Binary Data

```

<?xml version="1.0" encoding="utf-8"?>
<definitions
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  targetNamespace="urn:attachment.tip"
  xmlns:tns="urn:attachment.tip">
  <message name="empty"/>
  <message name="imageMsg">
    <part name="image" type="xsd:hexBinary"/>
  </message>
  <message name="octetMsg">
    <part name="octet" type="xsd:hexBinary"/>
  </message>

  <portType name="AttachmentTip">
    <operation name="sendImage">
      <input message="tns:imageMsg"/>
      <output message="tns:empty"/>
    </operation>
    <operation name="sendOctet">
      <input message="tns:octetMsg"/>
      <output message="tns:empty"/>
    </operation>
  </portType>

```



...



Binary Data *(Continued)*

```

<binding name="AttachmentBinding" type="tns:AttachmentTip">
  <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="sendImage">
    <soap:operation soapAction=""/>
    <input>
      <mime:multipartRelated>
        <mime:part>
          <soap:body use="literal"/>
        </mime:part>
        <mime:part>
          <mime:content part="image" type="image/jpeg"/>
        </mime:part>
      </mime:multipartRelated>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
  <operation name="sendOctet">
    <soap:operation soapAction=""/>
    <input>
      <mime:multipartRelated>
        <mime:part>
          <soap:body use="literal"/>
        </mime:part>
        <mime:part>
          <mime:content part="octet" type="application/octet-stream"/>
        </mime:part>
      </mime:multipartRelated>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>

```



Binary Data *(Continued)*

```
package tip.attachment;
```

```
import java.awt.Image;
```

```
import java.rmi.Remote;
```

```
import java.rmi.RemoteException;
```

```
import javax.activation.DataHandler;
```

```
public interface AttachmentTip extends Remote {  
    public void sendImage(Image image) throws RemoteException;  
    public void sendOctet(DataHandler octet) throws RemoteException;  
}
```





Binary Data *(Continued)*

```
POST /AttachmentWar/services/ImageBag HTTP/1.0
Content-Type: multipart/related; type="text/xml"; start="<2610731F21D947E13065B31A8CD35AA9>";
      =_Part_0_45152005.1092322773390"                boundary="----
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: IBM WebServices/1.0
Host: 127.0.0.1
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 15559
```

```
-----=_Part_0_45152005.1092322773390
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: binary
Content-Id: <2610731F21D947E13065B31A8CD35AA9>
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soapenv:Body
  soapenc:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"><sendImage xmlns=""><image
  href="cid:1D6BB8F2604E6FB5C2F9D710FAA0B506" xmlns:apache="http://xml.apache.org/xml-
  soap"/></sendImage></soapenv:Body></soapenv:Envelope>
```

```
-----=_Part_0_45152005.1092322773390
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-Id: <1D6BB8F2604E6FB5C2F9D710FAA0B506>
```

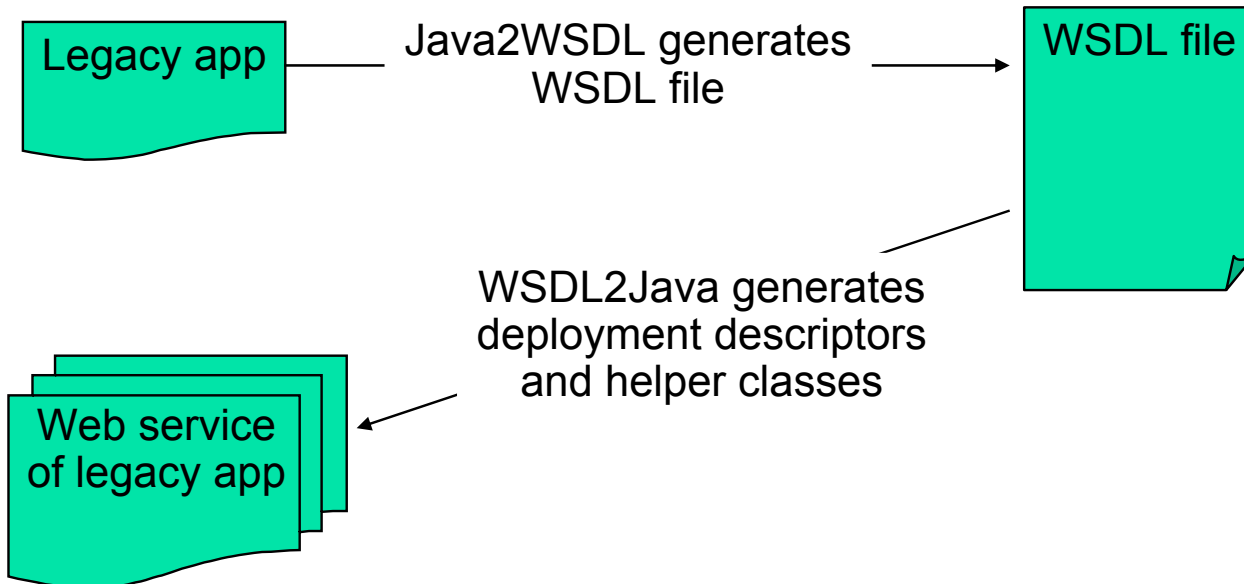
[binary image data]

```
-----=_Part_0_45152005.1092322773390--
```



Roundtripping Issues

Why do we care?





Roundtripping Issues *(Continued)*

- What is the problem?
 - java.util.Date → xsd:dateTime → java.util.Calendar
 - WSDL2Java forces Java coding conventions
- How is it solved?
 - Don't use java.util.Date
 - Always follow Java coding conventions
 - If you can't do these...
 - Write a Web services wrapper around the legacy app, or
 - The mapping meta-data file can fix many inconsistencies
 - (WAS 6.0 also provides an introspection mechanism)





Roundtripping Issues *(Continued)*

```
public interface population {  
    public life GetALife(java.util.Date date);  
}
```

||
v

Java2WSDL and WSDL2Java roundtrip

||
v

```
public interface Population {  
    public Life getALife(java.util.Calendar date);  
}
```





Exception Handling

- What is the problem?
 - Which of the many exceptions should an application throw?
 - `java.rmi.RemoteException`
 - `java.lang.RuntimeException`
 - `javax.xml.rpc.soap.SOAPFaultException`
 - checked user exceptions - mapped to/from `wsdl:fault`
- How is it solved?
 - We recommend user exceptions as much as possible





Array Gotcha

- What is the problem?
 - Null array *vs* empty array
 - Java makes a distinction
 - XML schema 'array' does not (minOccurs="0" maxOccurs="unbounded")
- How is it solved?
 - Use an array wrapper
 - Caveat: WSDL2Java does not necessarily map this to an array





Array Gotcha *(Continued)*

Typical schema with an 'array'

```
<complexType name="bean" >
  <sequence>
    <element name="name" type="xsd:string" />
    <element name="item" type="xsd:int"
      minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>
```

Instance – empty or null array?

```
<bean>
  <name>My bean</name>
</bean>
```





Array Gotcha *(Continued)*

Schema with a wrapped array

```
<complexType name="intArray" >
  <sequence>
    <element name="item" type="xsd:int" minOccurs="0"
      maxOccurs="unbounded" />
  </sequence>
</complexType>
<complexType name="bean" >
  <sequence>
    <element name="name" type="xsd:string" />
    <element name="array" type="intArray" nillable="true" />
  </sequence>
</complexType>
```

Instance – empty array

```
<bean>
  <name>My bean</name>
  <array />
</bean>
```

Software Summit



Web Resources

- IBM developerWorks web services domain
 - <http://www-106.ibm.com/developerworks/webservices/?loc=dwmain>
- IBM on-demand homepage
 - <http://www-3.ibm.com/e-business/index.html>
- IBM web services homepage
 - <http://www-4.ibm.com/software/solutions/webservices/>
- OASIS
 - <http://www.oasis-open.org/home/index.php>
- World Wide Web Consortium (W3C)
 - <http://www.w3.org/>
- Java Community Process (JCP)
 - <http://www.jcp.org>
- Apache
 - <http://xml.apache.org>
- WS-I
 - <http://ws-i.org>