seed
beyond the obvious

Chapter -

Sending and receiving complex data structures

Objectives

- At the end of this chapter you will be able to understand
  - How to send and receive complex data structures to and from a web service
  - Fault generation in a web service
  - Building a web service client from WSDL residing on a server

seed
beyond the obvious

Product query

- Suppose that our company would like to use web service to let our customers query the product availability and place orders with us.
- For this we need to discuss with them to decide on the interface. It doesn't make sense to say that *"When doing query, please send us an object of such a Java class. In this class there are this and that fields..."* because perhaps the people involved aren't programmers or don't use Java.
- Instead, XML is what is designed for this. It is platform neutral and programming language neutral
- So, suppose that we all agree on the following schema:

seed
beyond the obvious

Product query

Use the XML schema namespace as the default namespace. It defines elements such as <element>, <complexType> needed for you to define new elements.

Put your elements and types into this namespace

A <productQuery> contains one or more <queryItem> elements. Here is an example:

```
<?xml version="1.0"?>
<schema
   xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://foo.com">
   <element name="productQuery">
      <complexType>
         <sequence>
            <element name="queryItem" minOccurs="1" maxOccurs="unbounded">
               <complexType>
                  <attribute name="productId" type="string"/>
                  <attribute name="qty" type="int"/>
               </complexType>
            </element>
         </sequence>
      </complexType>
   </element>
</schema>
```

Define an element <productQuery>

The string type and int type are defined in the XML schema. They are usually shown as xsd:string and xsd:int, but the XML schema namespace here is the default namespace, so no prefix is needed.

A <queryItem> must appear at least once (1). There is no upper limit of its occurrence.

A <productQuery> has two attributes named "productId" and "qty" respectively.

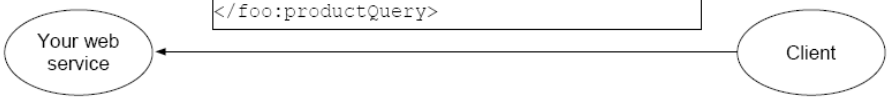
```
<?xml version="1.0"?>
<foo:productQuery xmlns:foo="http://foo.com">
   <queryItem productId="p01" qty="100"/>
   <queryItem productId="p02" qty="200"/>
   <queryItem productId="p03" qty="500"/>
</foo:productQuery>
```


seed
beyond the obvious

Product query

- When customers need to find out the availability of some products, they will send us a *<productQuery>* element.
- For example if they'd like to check if we have 100 pieces of p01, 200 pieces of p02 and 500 pieces of p03, they may send us a request like this:

```
<foo:productQuery
  xmlns:foo="http://foo.com">
  <queryItem productId="p01" qty="100"/>
  <queryItem productId="p02" qty="200"/>
  <queryItem productId="p03" qty="500"/>
</foo:productQuery>
```



Product query

- How does our web service reply??
- Use an XML element of course. So, in the schema we may have:

Product query

```
<?xml version="1.0"?>
<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://foo.com">
  <element name="productQuery">
    ...
  </element>
  <element name="productQueryResult">
    <complexType>
      <sequence>
        <element name="resultItem" minOccurs="1" maxOccurs="unbounded">
          <complexType>
            <attribute name="productId" type="string"/>
            <attribute name="price" type="int"/>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

For each <queryItem>, if the product is available, create a <resultItem> telling the unit price.

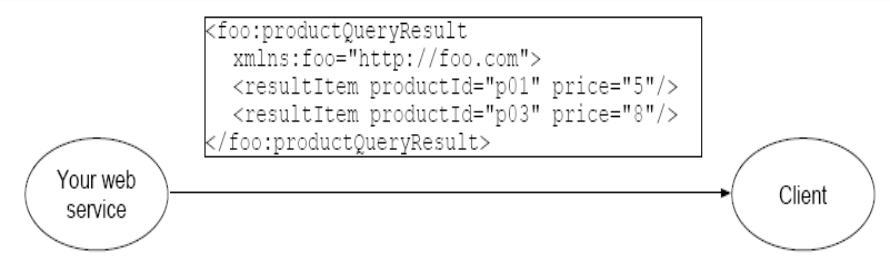
seed
beyond the obvious

Product query

- So, for the sample query above, if we have over 100 pieces of p01 and 500 pieces of p03 but only 150 pieces of p02, and we're willing to sell p01 at 5 dollars and p03 at 8 dollars each, we may reply:

seed
beyond the obvious

Product query



seed
beyond the obvious

Product query

- So what will be the interface for our web service??

seed
beyond the obvious

Product query-wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://foo.com"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="BizService"
 targetNamespace="http://foo.com">
 <wsdl:types>
   <xsd:schema targetNamespace="http://foo.com"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <xsd:element name="productQuery">
       <xsd:complexType>
         <xsd:sequence>
           <xsd:element name="queryItem" maxOccurs="unbounded" minOccurs="1">
             <xsd:complexType>
               <xsd:attribute name="productId" type="xsd:string">
               </xsd:attribute>
               <xsd:attribute name="qty" type="xsd:int">
               </xsd:attribute>
             </xsd:complexType>
           </xsd:element>
         </xsd:sequence>
       </xsd:complexType>
     </xsd:element>
```


seed
beyond the obvious

Product query-wsdl

```
    <xsd:element name="productQueryResult">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="resultItem" maxOccurs="unbounded" minOccurs="1">
            <xsd:complexType>
              <xsd:attribute name="productId" type="xsd:string">
              </xsd:attribute>
              <xsd:attribute name="price" type="xsd:int">
              </xsd:attribute>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
</wsdl:types>
```


seed
beyond the obvious

Product query-wsdl

```
<wsdl:message name="queryRequest">
  <wsdl:part name="parameters" element="tns:productQuery" />
</wsdl:message>
<wsdl:message name="queryResponse">
  <wsdl:part name="parameters" element="tns:productQueryResult" />
</wsdl:message>
<wsdl:portType name="BizService">
  <wsdl:operation name="query">
    <wsdl:input message="tns:queryRequest" />
    <wsdl:output message="tns:queryResponse" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="BizServiceSOAP" type="tns:BizService">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="query">
    <soap:operation soapAction="http://foo.com/NewOperation" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```


seed
beyond the obvious

Product query-wsdl

```
  <wsdl:service name="BizService">
    <wsdl:port binding="tns:BizServiceSOAP" name="BizServiceSOAP">
      <soap:address
        location="http://localhost:8080/axis2/services/BizService" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```


seed
beyond the obvious

Product query- Service stub and client stub

- Generate the service stub and client stub.
- Create a *BizServiceImpl* class in the *com.ttdev.biz* package:

seed
beyond the obvious

Product query- Service implementation

Let Eclipse add the unimplemented methods.

```
public class BizServiceImpl implements BizServiceSkeletonInterface {
    public ProductQueryResult query(ProductQuery productQuery) {

    }
}
```

XML elements are mapped to Java classes

```
<xsd:schema ...>
  <xsd:element name="productQuery">
    ...
  </xsd:element>
  <xsd:element name="productQueryResult">
    ...
  </xsd:element>
</xsd:schema>
```


seed
beyond the obvious

Product query – class mappings

- If we inspect the ProductQuery class and the *ProductQueryResult* class, we'll note the mapping is like this:

seed
beyond the obvious

Product query - class mappings

Each element in a sequence is mapped to a field in the class

An element that can occur multiple times (i.e., maxOccurs > 1) is mapped to a Java array

```
class ProductQuery {
   QueryItem_type0[] queryItem;
}
class QueryItem_type0
   String productId;
   int qty;
}
class ProductQueryResult {
   ProductQueryItem_type0[] resultItem;
}
class ResultItem_type0 {
   String productId;
   int price;
}
```

```
<xsd:element name="productQuery">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element
        name="queryItem"
        maxOccurs="unbounded"
        minOccurs="1">
        ...attr productId, qty...
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="productQueryResult">
    ...
</xsd:element>
```

Attributes are also mapped to fields, just like elements in a sequence.

The type for <queryItem> is mapped to a Java class. But why the class is not simply named QueryItem? Why the _type0 suffix? This is because the element is a local element and therefore it is possible for another top level element to contain another local <queryItem> element as shown below. Then that could be mapped to QueryItem_type1.

```
<xsd:element name="userQuery">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="queryItem"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```


seed
beyond the obvious

Product query- Service implementation class

- Fill in the code to complete the implementation and then deploy the service:

```
public class BizServiceImpl implements BizServiceSkeletonInterface {
  public ProductQueryResult query(ProductQuery productQuery) {
    ProductQueryResult result = new ProductQueryResult();
    QueryItem_type0[] queryItems = productQuery.getQueryItem();
    for (int i = 0; i < queryItems.length; i++) {
      QueryItem_type0 queryItem = queryItems[i];
      if (queryItem.getQty() <= 200) {
        ResultItem_type0 resultItem = new ResultItem_type0();
        resultItem.setProductId(queryItem.getProductId());
        resultItem.setPrice(20);
        result.addResultItem(resultItem);
      }
    }
    return result;
  }
}
```

Loop through each query item. Assume it's available if qty is <= 200.

Assume the unit price is always 20

seed beyond the obvious

Product query- Client class

- Create a BizClient.java:

```
public class BizClient {
  public static void main(String[] args) throws RemoteException {
    BizServiceStub bizService = new BizServiceStub();
    ProductQuery query = new ProductQuery();
    QueryItem_type0 queryItem = new QueryItem_type0();
    queryItem.setProductId("p01");
    queryItem.setQty(100);
    query.addQueryItem(queryItem);
    queryItem = new QueryItem_type0();
    queryItem.setProductId("p02");
    queryItem.setQty(200);
    query.addQueryItem(queryItem);
    queryItem = new QueryItem_type0();
    queryItem.setProductId("p03");
    queryItem.setQty(500);
    query.addQueryItem(queryItem);
    ProductQueryResult result = bizService.query(query);
    for (ResultItem_type0 resultItem : result.getResultItem()) {
      System.out.println(resultItem.getProductId() + ": "
          + resultItem.getPrice());
    }
  }
}
```


seed beyond the obvious

Product query- Client class

- Run the client and it should work:

```
Console  Problems  @ Javadoc  Declaration
<terminated> BizClient [Java Application] C:\Program Files\Java\jr
log4j:WARN No appenders could be found
log4j:WARN Please initialize the log4j
p01: 20
p02: 20
```


seed
beyond the obvious

Returning faults

- Suppose that a client is calling our query operation but a product id is invalid (not just out of stock, but absolutely unknown) or the quantity is zero or negative. We may want to throw an exception. To return an exception to the client, we send a "fault message", which is very much like an output message.
- To do that, modify the WSDL file:

seed
beyond the obvious

Returning faults- wsdl modifications

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions ...>
   <wsdl:types>
      <xsd:schema ...>
         <xsd:element name="productQuery">
            ...
         </xsd:element>
         <xsd:element name="productQueryResult">
            ...
         </xsd:element>
         <xsd:complexType name="queryItemType">
            ...
         </xsd:complexType>
         <xsd:element name="invalidProductId" type="xsd:string" />
         <xsd:element name="invalidQty" type="xsd:int "/>
      </xsd:schema>
   </wsdl:types>
   <wsdl:message name="queryRequest">
      <wsdl:part name="parameters" element="tns:productQuery" />
   </wsdl:message>
   <wsdl:message name="queryResponse">
      <wsdl:part name="parameters" element="tns:productQueryResult" />
   </wsdl:message>
   <wsdl:message name="queryInvalidProductId">
      <wsdl:part name="parameters" element="tns:invalidProductId" />
   </wsdl:message>
   <wsdl:message name="queryInvalidQty">
      <wsdl:part name="parameters" element="tns:invalidQty" />
   </wsdl:message>
   <wsdl:portType name="BizService">
      <wsdl:operation name="query">
         <wsdl:input message="tns:queryRequest" />
         <wsdl:output message="tns:queryResponse" />
         <wsdl:fault name="f01" message="tns:queryInvalidProductId" />
         <wsdl:fault name="f02" message="tns:queryInvalidQty" />
      </wsdl:operation>
   </wsdl:portType>
   ...
</wsdl:definitions>
```

The one and only part is a well defined element in the schema

A fault message is like an output message, but it indicates an error.

Unlike an input or output message which doesn't need a name, a fault needs a unique name because there can be multiple fault messages (here you have 2). Later you'll refer to a fault using its name.

seed beyond the obvious

Returning faults- wsdl modifications

- How to include the fault message in a SOAP message??
- It is included in the SOAP body, but not directly:

seed beyond the obvious

Returning faults- wsdl modifications

```
<wsdl:definitions ...>
  ...
  <wsdl:portType name="BizService">
    <wsdl:operation name="query">
      <wsdl:input message="tns:queryRequest" />
      <wsdl:output message="tns:queryResponse" />
      <wsdl:fault name="f01" message="tns:queryInvalidProductId" />
      <wsdl:fault name="f02" message="tns:queryInvalidQty" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="BizServiceSOAP" type="tns:BizService">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="query">
      <soap:operation soapAction="http://foo.com/NewOperation" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
      <wsdl:fault name="f01" >
        <soap:fault name="f01" use="literal"/>
      </wsdl:fault>
      <wsdl:fault name="f02" >
        <soap:fault name="f02" use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
  ...
</wsdl:definitions>
```

How to store this fault message in a binding?

In SOAP, include the fault message into the SOAP <Fault>:

The message part is already in XML

```
<soap-env:Envelope
  xmlns:soap-env="http://http://schemas.xmlsoap.org/soap/envelope/">
  <soap-env:Header>
    ...
  </soap-env:Header>
  <soap-env:Body>
    <soap-env:Fault>
      <soap-env:faultcode>...</soap-env:faultcode>
      <soap-env:faultstring>...</soap-env:faultstring>
      <soap-env:detail>
        <foo:invalidProductId xmlns:foo="http://foo.com">
          p1000
        </foo:invalidProductId>
      </soap-env:detail>
    </soap-env:Fault>
  </soap-env:Body>
<soap-env:Envelope>
```

seed beyond the obvious

Returning faults

- The SOAP <*Fault*> element tells the caller that something is wrong.
  - The <*faultcode*> is a QName acting as an error code.
  - The <*faultstring*> is an error message for human reading.
  - The <*detail*> will contain any information that both sides agree on. In this case, it contains our fault message part.

seed beyond the obvious

Returning faults

- Now, generate the service and client stubs and refresh the files in Eclipse. We will find some new Java classes:

seed
beyond the obvious

Returning faults- Fault message mapping

```
class QueryInvalidProductId extends Exception {
  InvalidProductId faultMessage;
  ...
}
```

A fault message is mapped to a Java exception. Its one and only part (an XML element) is mapped to a field.

As usual, an XML element such as the <invalidProductId> element is mapped to a Java class. It wanted to extend String, but String is a final class. So the string is mapped to a field.

```
class InvalidProductId {
  String invalidProductId;
  ...
}
```

```
class QueryInvalidQty extends Exception {
  InvalidQty faultMessage;
  ...
}
```

```
class InvalidQty {
  int invalidQty;
  ...
}
```


seed
beyond the obvious

Returning faults – Skeleton interface

- The method signature in *BizServiceSkeletonInterface* has also been updated to throw such exceptions:

```
public interface BizServiceSkeletonInterface {
    public ProductQueryResult query(ProductQuery productQuery)
    throws QueryInvalidProductId, QueryInvalidQty;
}
```

- Now modify our service implementation code accordingly:

seed
beyond the obvious

Returning faults – Service implementation class

```
public class BizServiceImpl implements BizServiceSkeletonInterface {
  public ProductQueryResult query(ProductQuery productQuery)
      throws QueryInvalidProductId, QueryInvalidQty {
    ProductQueryResult result = new ProductQueryResult();
    QueryItemType[] queryItems = productQuery.getQueryItem();
    for (int i = 0; i < queryItems.length; i++) {
      QueryItemType queryItem = queryItems[i];
      if (!queryItem.getProductId().startsWith("p")) {
        QueryInvalidProductId fault = new QueryInvalidProductId();
        InvalidProductId part = new InvalidProductId();
        part.setInvalidProductId(queryItem.getProductId());
        fault.setFaultMessage(part);
        throw fault;
      }
      if (queryItem.getQty() <= 0) {
        QueryInvalidQty fault = new QueryInvalidQty();
        InvalidQty part = new InvalidQty();
        part.setInvalidQty(queryItem.getQty());
        fault.setFaultMessage(part);
        throw fault;
      }
      if (queryItem.getQty() <= 200) {
        ResultItem_type0 resultItem = new ResultItem_type0();
        resultItem.setProductId(queryItem.getProductId());
        resultItem.setPrice(20);
        result.addResultItem(resultItem);
      }
    }
    return result;
  }
}
```


seed
beyond the obvious

Returning faults – Service client

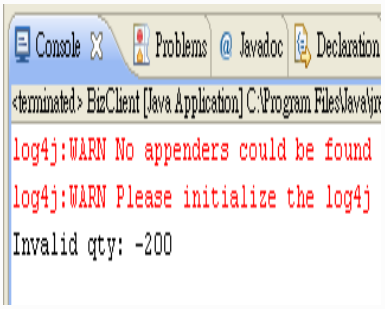
- To see if it's working, modify BizClient.java:

```
    public static void main(String[] args) throws RemoteException {
        BizServiceStub bizService = new BizServiceStub();
        ProductQuery query = new ProductQuery();
        QueryItemType queryItem = new QueryItemType();
        queryItem.setProductId("p01");
        queryItem.setQty(100);
        query.addQueryItem(queryItem);
        queryItem = new QueryItemType();
        queryItem.setProductId("p02");
        queryItem.setQty(-200);
        query.addQueryItem(queryItem);
        queryItem = new QueryItemType();
        queryItem.setProductId("p03");
        queryItem.setQty(500);
        query.addQueryItem(queryItem);
    try {
      ProductQueryResult result = bizService.query(query);
          for (ResultItem_type0 resultItem : result.getResultItem()) {
              System.out.println(resultItem.getProductId() + ": " +
                  resultItem.getPrice());
          }
    } catch (QueryInvalidProductId e) {
      System.out.println("Invalid product id: "
              + e.getFaultMessage().getInvalidProductId());
    } catch (QueryInvalidQty e) {
      System.out.println("Invalid qty: "
              + e.getFaultMessage().getInvalidQty());
    }
  }
}
```


seed
beyond the obvious

Returning faults – Service client

- Run the BizClient and it should work:




```
<?xml version='1.0' encoding='UTF-8'?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
      <soapenv:Fault>
        <faultcode>soapenv:Server</faultcode>
        <faultstring>QueryInvalidQty</faultstring>
        <detail>
          <ns1:invalidQty xmlns:ns1="http://foo.com">-200</ns1:invalidQty>
        </detail>
      </soapenv:Fault>
    </soapenv:Body>
  </soapenv:Envelope>
```


seed
beyond the obvious

Retrieving WSDL files using HTTP

- To really simulate the client side, it should retrieve the WSDL file using *http://localhost:8080/axis2/services/BizService?wsdl* instead of a local file.
- To verify that, modify *build.xml*:

seed
beyond the obvious

Retrieving WSDL files using HTTP

```
<project ...>
  ...
  <target name="generate-client">
    <wsdl2code
      wsdlfilename="http://localhost:8080/axis2/services/BizService?wsdl"
      skipbuildxml="true"
      namespacetopackages="http://foo.com=com.ttdev.biz.client"
      targetsourcefolderlocation="src"
      overwrite="true"/>
  </target>
</project>
```


seed
beyond the obvious

Quick Recap . . .

- For better performance, we should design the interfaces of our web service operations so that more data is sent in a message.
- To report an error from our operation, define a message in the WSDL file and use it as a fault message in the operation. Then add a corresponding child element in the SOAP binding to store it into the SOAP Fault element. The fault message should contain one and only one part which is an XML element describing the fault. The <wsdl2code> Ant task will map a fault message to a Java exception class and the part as a field. The operation will be mapped to a Java method throwing that exception.
- We can generate the client by referring a wsdl file on the server

seed
beyond the obvious