Import of the XSD Element Content

The purchase order schema has many examples of elements containing other elements (e.g. items), elements having attributes and containing other elements (e.g. shipTo), and elements containing only a simple type of value (e.g. USPrice). However, we have not seen an element having attributes but containing only a simple type of value, nor have we seen an element that contains other elements mixed with character content, nor have we seen an element that has no content at all. In this section we will examine these variations in the content models of elements.

On this Page:

Complex Types from Simple Types

Let us first consider how to declare an element that has an attribute and contains a simple value. In an instance document, such an element might appear as:

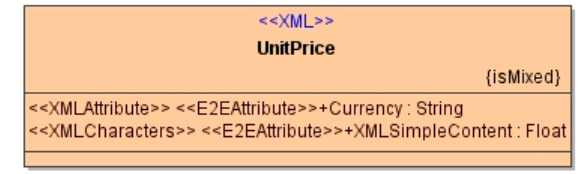
```
<QuantityPrice Currency="EUR">423.46</QuantityPrice>
```

As we have said before, simple types cannot have attributes, so **QuantityPrice** cannot be declared as **xs d:decimal**. Therefore, we must define a complex type to carry the attribute declaration. We also want the content to be simple type decimal. So, how do we define a complex type that is based on the simple decimal type? The answer is to extend a new complex type from the simple type:

Figure: Deriving a Complex Schema Type From a Simple One

```
<xsd:element name="UnitPrice"
type="UnitPrice" >
        <xsd:complexType name="
UnitPrice">
                  <xsd:
simpleContent>
                    <xsd:
extension base="xsd:decimal">

<xsd:attribute name="
Currency"  type="xsd:string"/>
                     </xsd:
extension>
                  </xsd:
simpleContent>
        </xsd:complexType>
```



In XML Schemas, we use the **complexType** element to start the definition of a new type. To indicate that the content model of the new type contains only character data and no elements, we use a **simpleConte nt** element. Finally, we derive the new type by extending the simple decimal type. The **extension** consists of adding a **Currency** attribute using a standard attribute declaration.

The importer creates a class **UnitPrice** class corresponding to the complex type. Note, the XSD extension does not correspond to an UML generalization. Because if it did, the class **UnitPrice** would derive from the simple UML type **Float** which would make it a simple type as well. Which is a contradiction because simple types cannot hold attributes. Instead, the simple content of the above element (423.46) is stored in the UML attribute **XMLSimpleContent**. The importer will assign the stereotype <<XMLCharacters>> to this attribute. Additionally, the importer sets the tagged value **isMixed** to true. Otherwise, the XML parser would not parse the XML instances correctly.

The general mapping rule for extending simple to complex types is:

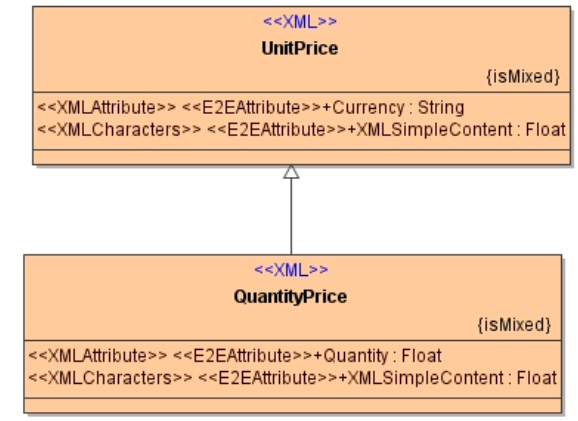
> **Mapping Rule:** Complex schema types having a **simpleContent** content model map to UML classes that contain exactly one <<XMLCharacters>> **XMLSimpleContent** attribute. The XSD extension does not map to an UML generalization if the complex type extends a simple type.

An example where the extension maps to a generalization is found in the following table, where the complex type **QuantityPrice** extends **UnitPrice** with an additional attribute **Quantity**:

Figure: Deriving a simpleContent Complex Type From Another simpleContent Complex Type

```
<xsd:complexType name="
QuantityPrice">
          <xsd:
simpleContent>
            <xsd:
extension base="
UnitPrice">

<xsd:attribute name="
Quantity"  type="xsd:
decimal"/>
             </xsd:
extension>
          </xsd:
simpleContent>
</xsd:complexType>
```



Mixed Content

The construction of the purchase order schema may be characterized as elements containing subelements, and the deepest subelements contain character data. XML Schema also provides for the construction of schemas where character data can appear alongside subelements, and character data is not confined to the deepest subelements.

To illustrate, consider the following snippet from a customer letter that uses some of the same elements as the purchase order:

Figure: Snippet of customer letter

```
<letterBody>
          <salutation>Dear Mr.<name>Robert Smith</name>.</salutation>
          Your order of <quantity>1</quantity>
          <productName>Baby Monitor</productName> shipped from our
          warehouse on
          <shipDate>1999-05-21</shipDate>. ....
</letterBody>
```

Notice the text appearing between elements and their child elements. Specifically, text appears between the elements salutation, quantity, productName, and shipDate which are all children of letterBody, and text appears around the element name which is the child of a child of letterBody. The following snippet of a schema declares letterBody:

Figure: Snippet of schema for customer letter

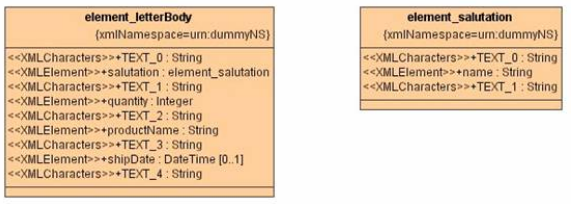
```
<xsd:element name="letterBody">
          <xsd:complexType mixed="true">
             <xsd:sequence>
               <xsd:element name="salutation">
                       <xsd:complexType mixed="true">
                               <xsd:sequence>
                                       <xsd:element name="name" type="xsd:
string"/>
                           </xsd:sequence>
                           </xsd:complexType>
                    </xsd:element>
               <xsd:element name="quantity" type="xsd:positiveInteger"/>
               <xsd:element name="productName" type="xsd:string"/>
               <xsd:element name="shipDate" type="xsd:date" minOccurs="0"
/>
               <!-- etc. -->
             </xsd:sequence>
          </xsd:complexType>
</xsd:element>
```

The elements appearing in the customer letter are declared, and their types are defined using the element and complexType element constructions we have seen before. To enable character data to appear between the child-elements of letterBody, the mixed attribute on the type definition is set to true. The Importer take such a mixed content model and generates UML classes following the

> **Mapping Rule:** All UML attributes corresponding to an XML element are wrapped by UML attributes of stereotype <<XMLCharacters>> .

The following picture illustrates this:

Figure: Result of importing a mixed content model



If one would instantiate above classes using the example at the beginning of this section one would get the following attribute values:

```
Element_letterBody.TEXT_0 = ""
Element_letterBody.salutation.TEXT_0 = "Dear Mr."
Element_letterBody.salutation.name = "Robert Smith"
Element_letterBody.salutation.TEXT_1 = "."
Element_letterBody.TEXT_1 = "Your order of "
Element_letterBody.quantity = 1
Element_letterBody.TEXT_2 = ""
Element_letterBody. productName = "Baby Monitor"
Element_letterBody.TEXT_3 = "shipped from our warehouse on"
Element_letterBody..shipDate = "1999-05-21"
Element_letterBody.TEXT_4 = ""
```

anyType

The anyType represents an abstraction called the ur-type which is the base type from which all simple and complex types are derived. An anyType type does not constrain its content in any way. Thus, the WSDL Importer does not support type references to anyType and prints an error message if it finds such references.

However, the Importer does support anyTypes as base types of restrictions, like in the following example:

Figure: AnyType as base type

```
<xsd:element name="element1">
          <xsd:complexType>
             <xsd:complexContent>
               <xsd:restriction base="xsd:anyType">
                         <xsd:attribute name="currency" type="xsd:string"
/>
                   <xsd:attribute name="value" type="xsd:decimal"/>
               </xsd:restriction>
             </xsd:complexContent>
          </xsd:complexType>
</xsd:element>
```

The above element is equivalent to the following XML Schema:

Figure: Simplified representation: anyType as base type

```
<xsd:element name="element2">
           <xsd:complexType>
             <xsd:attribute name="currency" type="xsd:string"/>
              <xsd:attribute name="value" type="xsd:decimal"/>
           </xsd:complexType>
</xsd:element>
```

It is the latter XML Schema that does not refer to the anyType that gets actually imported.