Import of Simple XSD Types

The purchase order schema declares several elements and attributes that have simple types. Some of these simple types, such as string and decimal, are built in to XML Schema, while others are derived from the built-ins. For example, the partNum attribute has a type called SKU (Stock Keeping Unit) that is derived from string. Both built-in simple types and their derivations can be used in all element and attribute declarations. The following table lists all the simple types built in to XML Schema, along with the corresponding internal types.

On this Page:

Find below a table of the built-in XSD types and their E2E Bridge counterparts.

Mapping Rule	
XSD Type	**Internal Type**

`duration`
`dateTime`
`time`
`date`
`gYearMonth`
`gYear`
`gMonthDay`
`gDay`
`gMonth`
	DateTime

`boolean`	Boolean
`base64Binary` `hexBinary`	Blob
`float` `double`	Float

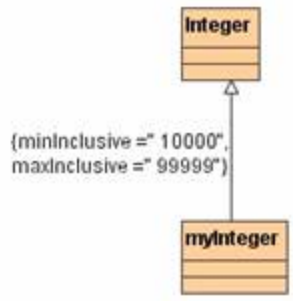
`anyURI`
`QName`
`NOTATION`
`string`
`normalizedString`
`token`
`language`
`Name`
`NCName`
`ID`
`IDREF`
`IDREFS`
`ENTITY`
`ENTITIES`
`NMTOKEN`
`NMTOKENS`
	String

`decimal`
`integer`
`nonPositiveInteger`
`negativeInteger`
`long`
`int`
`short`
`byte`
`nonNegativeInteger`
`unsignedLong`
`unsignedInt`
`unsignedShort`
`unsignedByte`
`positiveInteger`
	Integer

In XML Schemas, new simple types are defined by deriving them from existing simple types (built-ins and derived). In particular, we can derive a new simple type by restricting an existing simple type, in other words, the legal range of values for the new type are a subset of the existing type's range of values. This can be done in UML as well as the next table illustrates:

Figure: Deriving user defined simple types from built-in types

```
<xsd:simpleType name="myInteger">
        <xsd:restriction base="xsd:integer">
            <xsd:minInclusive value="10000"/>
              <xsd:maxInclusive value="99999"
/>
           </xsd:restriction>
</xsd:simpleType>
```



Generally speaking, the following rule emerges:

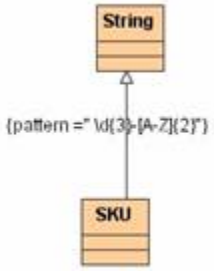
Mapping Rule: xsd:restriction corresponds to an UML generalization relationship having an UML constraint. The constraint value equals the contents of the restriction element translated into name value pairs

These constraints are not OCL (Object Constraint Language) compliant.

The viable constraint names are found in the following XML Schema 1.0 specification, appendix B. The purchase order schema contains another, more elaborate, example of a simple type definition. A new simple type called SKU is derived (by restriction) from the simple type string. Furthermore, we constrain the values of SKU using a facet called pattern in conjunction with the regular expression "\d{3}-[A-Z]{2}" that is read "three digits followed by a hyphen followed by two upper-case ASCII letters":

Figure: Deriving new a new string type using a pattern

```
<xsd:simpleType name ="SKU">
        <xsd:restriction base ="xsd:string">
              <xsd:pattern value ="\d{3}-[A-Z]{2}"
/>
    </xsd:restriction >
</xsd:simpleType >
```

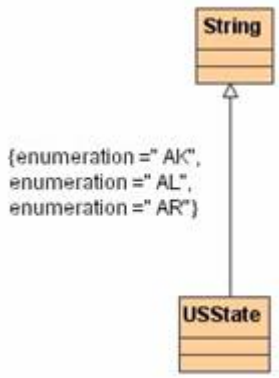


This regular expression language is described more fully in the XML Schema 1.0 specificaton, appendix D.

XML Schema defines fifteen constraints which are listed in appendix B. Among these, the enumeration fa cet is particularly useful and it can be used to constrain the values of almost every simple type. For example, we can use the enumeration facet to define a new simple type called USState, derived from string, whose value must be one of the standard US state abbreviations:

Figure: Applying enumeration constraints

```
<xsd:simpleType name ="USState">
        <xsd:restriction base ="xsd:string">
              <xsd:enumeration value ="AK"/>
              <xsd:enumeration value ="AL"/>
              <xsd:enumeration value ="AR"/>
      <!-- and so on ... -->
           </xsd:restriction >
</xsd:simpleType >
```



List Types

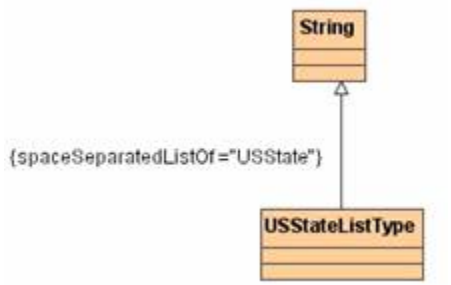
XML Schema has the concept of a list type, in addition to the so-called atomic types that listed in Figure 433. The value of an atomic type is indivisible from XML Schema's perspective. For example, the NMTOKEN value US is indivisible in the sense that no part of "US", such as the character "S", has any meaning by itself. In contrast, list types are comprised of sequences of atomic types and consequently the parts of a sequence (the "atoms") themselves are meaningful. For example, NMTOKENS is a list type, and an element of this type would be a white-space delimited list of NMTOKEN's, such as "US UK FR". XML Schema has three built-in list types, they are NMTOKENS, IDREFS, and ENTITIES.

Note that it is possible to derive a list type from the atomic type string. However, a string may contain white space, and white space delimits the items in a list type, so you should be careful using list types whose base type is string.
However, the Bridge does not support this kind of lists but follows the mapping rule: xsd:list types are mapped to Bridge Base Types::Strings.

The following example shows the XML Schema definition of a list of US states and the corresponding UML classes:

Figure: Example of an XSD list type

```
<xsd:simpleType name ="
USStateListType">
        <xsd:list itemType ="
USState"/>
</xsd:simpleType >
```



Above table shows that by mapping the list type to string. The constraint holds the information the USStateListType is actually a space separated list of USStates. If the modeler wants to decompose such a string into an array of its components she can use the split operation applying regular expression to generate such a list. This enables her also to take into account atomic types that contain spaces.

Union Types

Atomic types and list types enable an element or an attribute value to be one or more instances of one atomic type. In contrast, a union type enables an element or attribute value to be one or more instances of one type drawn from the union of multiple atomic and list types. To illustrate, we create a union type for representing American states as singleton letter abbreviations or lists of numeric codes. The zipUnion union type is built from one atomic type and one list type:

Figure: Example of an XSD union type

```
<xsd:simpleType name ="zipUnion">
          <xsd:union memberTypes ="
USState listOfMyIntType"/>
</xsd:simpleType >
```

Again, the Bridge does not support unions, therefore using the same rule as for list types:

Mapping Rule: xsd:union types are mapped to E2E Bridge Base Types::Strings.

The conversion to string is done by the E2E Runtime. If the conversion is not possible, a runtime error occurs.