XML - UML Class Mapping

The following operations and components are using a built-in mapping of XML to UML classes and vice versa:

- Components:
  - SOAP Adapter and SOAP Service exchanging **document style/literal encoded** messages.
    The mapping procedures are used, when mapping SOAP messages to UML classes and back.

    > The mapping procedures described below do not work with SOAP/RPC encoding.

  - REST Adapter and REST Service
    The mapping procedures are used, when mapping REST calls to UML classes and back.
- Operations:
  - xmlToClass() Operation for Strings
  - xmlToClass() Operation for Blobs
  - classToXML()
  - classToJSON() Operation
  - jsonToClass() Operation

Mapping of UML classes to XML documents is controlled by stereotypes assigned to class attributes and association ends.

Example File (Builder project E2E Action Language/XML):

<your example path>\E2E Action Language\XML\uml\xmlSimpleConversions.xml
<your example path>\E2E Action Language\XML\uml\xmlComplexConversions.xml

Example File (Builder project Add-ons/URL):

<your example path>\Add-ons\URL\uml\urlUrl.xml

On this Page:

- Controlling the Mapping by Using Stereotypes
  - Stereotypes
  - Tagged Values
- Examples
  - Mapping Mixed Content XML to a UML Class
  - Arrays and XML Serialization

Related Pages:

- xmlToClass()
- classToXML()
- Troubleshooting XML - UML Class Mapping
- Number Formatting
- Date and Time Formatting
- Importing WSDL or XSD
- XML Schema Import Rules pp.

Controlling the Mapping by Using Stereotypes

Stereotypes control how UML class properties (attributes and association ends) are serialized to an XML document. If no stereotype is assigned, the following default rules apply for mapping XML to UML classes:

- XML attributes are mapped to class attributes.
- XML elements are mapped to class associations ends.
- The XML root element is named according to the definitions in the service composite, tagged value **Class To XML Default Root Name** (also see Frontend Components).

Stereotypes

To control the mapping, the following stereotypes are available:

Stereotype	Description	Tagged Values (Description see below)
Class		

<<XML>>	Set this basic stereotype on class level to be able to apply the stereotypes listed below to the attributes. You can only apply one of these stereotypes at a time.	**xmlNamespace** **xmlElementName** **isMixed** **isOrdered**
Attribute		
All stereotypes listed below inherit from <<E2EAttribute>> which gives additional tagged values. In this context of XML-UML mapping, only **externalName** and **order** are relevant.		**externalName** **order**
<<XMLNamespace>>	When this stereotype is applied to an UML property, it will be mapped to a XML namespace. The property must be of base type **String**. • The prefix of the namespace is given by the property name. • The URI of the namespace is given in the property value. • If the same namespace is declared more than once, the runtime will suppress namespaces further down the XML hierarchy, if prefix and namespace are identical. If not, the xUML Runtime will throw an exception.	
<<XMLElement>>	When this stereotype is applied to an UML property, it will be mapped to an XML element.	**xmlNamespace** **xmlForm** **xmlFormat** **xmlArrayElement** **isNillable**
<<XMLAttribute>>	When this stereotype is applied to an UML property, it will be mapped to an XML attribute. The property must be of simple type. If the property type is complex, the compiler will report an error.	**xmlNamespace** **xmlForm** **xmlFormat**
<<XMLCharacters>>	When this stereotype is applied to an UML property, its content will be mapped to a character stream. The property must be of simple type.	**xmlFormat**

Tagged Values

Tagged Value	Stereotype	Description	Allowed Values	
Classes				
classToXMLDefaultRootName	<<E2EComposite>>	Bridge 7 Specify which name to assign to the XML root element upon serializing. This setting can be overridden by using XML composer options as described on classToXML() Operation. If this tagged value is set to other values than "Default", tagged values **xmlElementName** and **xmlNamespace** on the <<XML>> class are disregarded.	**Default**	Try to use name and namespace defined on the class by the <<XML>> stereotype. Fallback to **Variable Name** if not provided (default).
			Type Name	Use static name and namespace of the class as name of XML root element.
			Variable Name	Use the name of the reference (object /variable) as name of XML root element.
xmlNamespace	<<XML>>	Specify the XML namespace.	a valid namespace	
xmlElementName	<<XML>>	Specify the name of the XML root element.	a valid element name	

isMixed	<<XML>>	Specify whether the XML contains attributes that are serialized as character stream (attributes with stereotype <<XMLCharacters>>, see also Controlling the Mapping by Stereotypes). For more information on mixed content, refer to Mixed Content.	true	XML contains serialized attributes.
			false	XML does not contain serialized attributes (default).
isOrdered	<<XML>>	Specify whether the class attributes should be serialized to XML using the **order** tag that has been specified on the attributes.	true	Serialize in order of **order** tags from the attributes.
			false	Serialize in order of attributes on class (default).
Attributes				
xmlNamespace	<<XMLElement>> <<XMLAttribute>>	Each XML attribute and element may have its own namespace. If the tagged value contains an URI, the runtime will automatically generate a unique prefix. Examples: • A tagged value **xmlNamespace** = "http://e2e.ch" on the UML property **anElement** will result in the XML document `<ns0:anElement xmlns:ns0="http://e2e.ch">`. • Also, it is possible to define the prefix by using the following syntax: `'xmlns:' <prefix name> '=' <namespace uri>`. For example, tagged value **xmlNamespace** = 'xmlns:typens="http://e2e.ch"' of the UML property **anElement** will lead to the following XML fragment: `<typens:anElement xmlns:typens="http://e2e.ch">`.	an URI	
			a valid xlmns syntax	
xmlForm	<<XMLElement>> <<XMLAttribute>>	Depending on this tagged value, XML elements or attributes may not be qualified by a namespace prefix even if they have one.	qualified	The element or attribute must always be qualified by a namespace prefix. **Default for XML elements.**
			unqualified	No namespace prefixes are allowed (for details see http://www.w3.org/TR/xmlschema-0/#NS). **Default for XML attributes.**
xmlFormat	<<XMLElement>> <<XMLAttribute>> <<XMLCharacters>>	If numbers and date/time types are parsed or composed, the XML parser respectively composer expects simple date types following the XML schema specification. However, legacy XML documents may contain different number and date/time formats. In this case, the tagged value **xmlFormat** may hold a format string. • If numbers are parsed or composed use the format strings defined in section Number Formatting. • If date/time expressions must be parsed or composed, use the format strings defined in Date and Time Formatting. Use **xmlFormat** = "CDATA" together with stereotype <<XMLElement>> to *compose* strings as CDATA with `classToXML()` . *Parsing* CDATA elements works out of the box, you do not need to set **xmlFormat**.	a valid format string (see Number Formatting or Date and Time Formatting)	
			CDATA	compose string as CDATA
isNillable	<<XMLElement>>	By default, UML properties that are NULL are not serialized into XML documents. However, if it is necessary to do so, **isNillable** must be set to true. In this case, the UML properties being NULL will look like: `<aProperty xsi:nil="true"></aProperty>`. Some client code generators will use this attribute in the WSDL file for type generation. If false they will generate simple types, if true they will generate complex types.	true	serialize NULL properties
			false (default)	do not serialize NULL properties
order	<<XMLNamespace>> <<XMLElement>> <<XMLAttribute>> <<XMLCharacters>>	Use **order** to specify the order in which the XML elements will be generated to the XML document.	a valid float	

externalName	<<XMLNamespace>> <<XMLElement>> <<XMLAttribute>> <<XMLCharacters>>	Sometimes names of XML attributes or elements do not comply with the rules specified on Syntax Scheme of the xUML Action Language. Use **externalName** to specify the name of the XML attribute or element, and specify a better name via the name of the UML property for internal usage.	

Examples

Mapping Mixed Content XML to a UML Class

Example File (Builder project Add-ons/URL):

<your example path>\Add-ons\URL\uml\urlUrl.xml

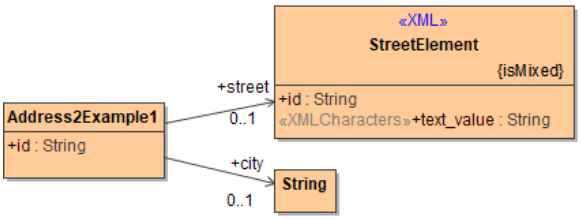
The following example shows how to convert an XML data structure with mixed content to an UML class.

> If an XML element has attributes, this is called **mixed content** in Bridge context.

Suppose the following XML data structure with two elements **street** and **city** is given:

```
<address id="4711">
        <street id="4711-1">13, Coal Street</street>
        <city id="4711-2">New York, NY 10017, USA</city>
</address>
```

The class diagram below illustrates how to map the content of XML element **street** ("13, Coal Street") and the value of its attribute **id** ("4711-1") to a class structure:



To map XML element **street** and its attribute **id**, you need a dedicated class as a container. In this example, this is class **StreetElement**. XML attribute **id** of element **street** is mapped to class attribute **id** of UML class **StreetElement** via matching names. The content of the XML element **street** is mapped to the class attribute that has stereotype <<XMLCharacters>>. In this example this is **text_value**.
Class **StreetElement** must have stereotype <<XML>> and tagged value **isMixed** set to true because it contains the serialized attribute **text_value** (also see explanation of tagged value **isMixed**).

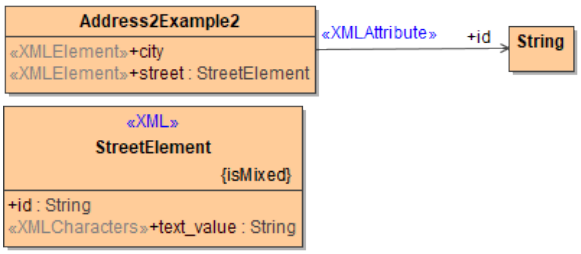
Containter **StreetElement** is associated to the main address class. The name of the association end (**street**) must match the name of the XML element. Associated class **StreetElement** is only a container for the street element and its name is not relevant for the mapping.

If the XML element contains attributes, which should not be mapped to UML class attributes, an association to a base type can be used. The example shows this for element **city** . The content of the element ("New York, NY 10017, USA") will be available in **city** (accessible as attribute of the instantiated class **Address2Example1**). As the id attribute cannot be mapped, it will be discarded when executing `xmlToClass()` .

If an object of class **Address2Example1** was serialized back to a SOAP response, the SOAP message would read:

```
<address2Example1 xmlns:ns1="urn:Services.URLService.URLPort.DataItems"
xsi:type="ns1:Address2Example1">
    <id xsi:type="xsd:string">4711</id>
    <street xsi:type="ns1:StreetElement">
              <id xsi:type="xsd:string">4711-1</id>
              <text_value xsi:type="xsd:string">13, Coal Street<
/text_value>
    </street>
    <city xsi:type="xsd:string">New York, NY 10017, USA</city>
</address2Example1>
```

The same result could be obtained with the following class diagram:



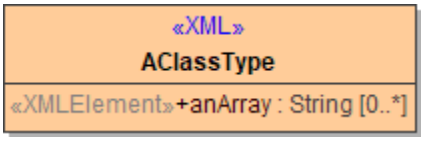
You have the flexibility to map XML structures in different ways. You can define very compact class diagrams for simple XML structures, but you have also the possibility to map complex XML data structures graphically.

> If an XML Schema is provided, you can import it with the Builder. It will generate all classes and relations according the XSD import rules described in XML Schema Import Rules. For how to use the XSD Importer refer to Importing WSDL or XSD.

Arrays and XML Serialization

The behavior of XML serialization in this case is not always self-explanatory but a consequence of the definition of arrays in XML schema.

The following table shows the behavior of XML serialization for the following class containing an array:




```
{
        "aClass" : { "anArray" : ["A1", "A2", "A3"] };
}
```

Description	XML result
All values are present.	<aClass> <anArray>A1</anArray> <anArray>A2</anArray> <anArray>A3</anArray> </aClass>

The second value (A2) is NULL and **isNillable**=false.	<aClass> <anArray>A1</anArray> <anArray>A3</anArray> </aClass>
The second value (A2) is NULL and **isNillable**=true	<aClass xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <anArray>A1</anArray> <anArray xsi:nil="true"/> <anArray>A3</anArray> </aClass>
The array is NULL.	<aClass> </aClass>
The array is empty.	<aClass> </aClass>
All elements are NULL and **isNillable**=false.	<aClass> </aClass>
All elements are NULL and **isNillable**=true.	<aClass xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <anArray xsi:nil="true"/> <anArray xsi:nil="true"/> <anArray xsi:nil="true"/> </aClass>
classToXMLDefaultRootName="Default" and **xmlElementName**="anotherClass".	<anotherClass> <anArray>A1</anArray> <anArray>A2</anArray> <anArray>A3</anArray> </anotherClass>
classToXMLDefaultRootName="Type Name" and **xmlElementName**="anotherClass".	<AClassType> <anArray>A1</anArray> <anArray>A2</anArray> <anArray>A3</anArray> </AClassType>
classToXMLDefaultRootName="Variable Name" and **xmlElementName**="anotherClass".	<aClass> <anArray>A1</anArray> <anArray>A2</anArray> <anArray>A3</anArray> </aClass>