classToXML

Syntax

```
set aBlob = anObject.classToXML()
set aBlob = anObject.classToXML(options)
```

Semantics

The operation takes any object (`anObject`) and tries to map it to an XML document. The document structure is given by the class definition of `anObject`. If this is not possible an error is raised (e.g. XML parser errors, invalid mappings, etc.).

By default the following mapping rules apply:

- Class attributes are mapped to XML attributes.
- Association ends are mapped to XML elements.

These default rules can be overridden by using <<XMLElement>>, <<XMLAttribute>> and <<XMLCharacters>> stereotypes on UML class attributes and UML association ends.
More about these mapping rules, stereotypes and tagged values (e.g. for number and date & time formatting) can be found at XML - UML Class Mapping.
Builder 7.5.0 Additionally, you can hide attributes from being serialized by applying stereotype <<E2EPrivate>>. Refer to Hiding Attributes From Interfaces for more information.

> The WSDL/XSD importer will not set these XML stereotypes, hence `classToXml()` without options will always render the root element after the variable name. Bridge 7 This behavior can be overruled by setting **Class To XML Default Root Name** on the composite (see Frontend Components for more information).

> A conversion with operation `classToXML()` always returns an object of type **Blob**. To display this data as a **String** you need to transcode it first (see chapter transcodeToString() Operation).

Substitutables

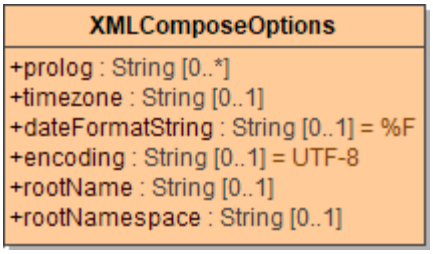
`anObject`	Target object can be any complex object. However, simple types and arrays are not supported, since they do not map naturally to a well formed XML document.

Related Pages:

- classToXMLFragment() Operation
- XML - UML Class Mapping
- Hiding Attributes From Interfaces
- transcodeToString() Operation

`options`

This optional parameter is an object of type **XMLComposeOptions**.



Its attributes are:

Attribute	Type	Description	Example
prolog	Array of String	The string values are inserted right before the root element of the generated document. This mechanism can be used to insert processing instructions (e.g. DTD and Schema references), comments, entities or any other prolog you may think of. However, be aware that using prolog arrays makes it easy to generate non-well-formed documents.	
timezone	String	Time zone string as specified in the time zone appendix. The timezone is used to print dateTime expressions. If no timezone is given, UTC is used. If "`local`" is used, the date/time is printed relative to the local timezone of the server, e.g. `2012-10-01T12:36:47.0+02:00` (the timezone of the server is UTC+02:00).	`"Australia/Melbourne"`, `"CET"`, `"Etc/GMT+10"`
dateFormatString	String	A format string to be used when printing DateTime values as `xs:date` (e.g. `%F` to print a date without timezone). The allowed formats can be found in time zone appendix. If nothing is defined, the XSD standard is used.	`%F`
encoding	String	Encoding of the target xml. Default encoding is **UTF-8**. For a list of possible encodings see Charset Definitions.	`"UTF-8"`
rootName	String	Bridge 7 Name of the generated XML root element. Use this tagged value to override the default behavior specified on the composite (see Frontend Components). Bridges of version 6 do not support this tagged value. The generated root element will be named after the variable name of the target object of the operation (`an0bject`). So, to influence the name of the root element, assign the desired name to the target object.	`TXTRAW01`
rootNamespace	String	Name of the namespace of the generated XML root element.	

Examples

```
set xmlBlob = myAddress.classToXML();
```

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

The following action script serializes an object of type **Address** provided, that you have defined an input object node named **myAddress** of type **Address** in the activity diagram.

```
set xmlBlob = myAddress.classToXML();
```

The sample XML document below illustrates the mapping executed by `classToXML()`. The object **myAddress** of type **Address** (see class diagram) is mapped to an XML document as depicted in the following XML document:

```
 <myAddress id="myAddressID">
   <street>Lautengartenstr. 12</street>
   <city>Basel</city>
</myAddress>
```

Note, that the XML element myAddress is of type **Address**. This type has the UML attribute **id**, which corresponds to the XML attribute id. Additionally, the XML elements street and city are mapped to the association ends **city** respectively **street**. Both are having the type **String**.