XML Binding from the DDI Model

1. Version 3 – 2015-05-27
2. Introduction

This document presents the mapping rules from the XML export from Drupal to the XML schema deliverables.

1. High-Level XML Constructs – XML schema files

For each library package in the model, we will have an XML namespace declared and an own schema file created. The XMI of the DDI model must place the name of the UML package (its URI) into the name attribute of the <packagedElement> which represents it. Configuration information will be provided at the time of the transformation. (This could be a parameter passed into an XSLT transformation, for example.)

* Small XML file will provide prefixes and identifiers for each importable namespace, for declaration in the root element of the schema, and for xsd:import statements. The transformation will identify which are needed, and input the appropriate xmlns:[prefix] declarations, and the correct import statements. The content will look like this:  
  <Namespace name="Core" prefix="ddic" location="ddi-core.xsd"/>  
  <Namespace name="Util" prefix="ddiu" location="ddi-util.xsd"/>  
  <Namespace name="Foundational" prefix="ddif" location="ddi-foundation.xsd"/>
* Packages are identified by <packagedElement xmi:type=”uml:Package”> that is a child node under <packagedElement xmi:id=”ddi4\_model”>. The name attribute is used to get the value of the xmlns attribute from the XML file mentioned above.
* In every library XML schema file we import all other library schema files, for mapping to namespace identifiers with xsi:schemaLocation. The namespace identifiers will exist in the model.
* The Classes from the package with the name “ComplexDataTypes” are handled slightly different. Details within the section “Class-Level Mappings”.
* For every view, that is part of the Drupal export as a package with references to the contained classes, again an XML schema file is created. A view is identified by <packagedElement xmi:type=”uml:Package”> that is a child node under <packagedElement xmi:id=”ddi4\_model”>. The name attribute is used to get the value of the xmlns attribute from the XML file mentioned above. The XML schema file contains one complex type definition for a root element named after the view, which can contain all the classes referenced. Therefor, all library schema definitions are imported. A distinction, whether an import is needed or not, is not made. A corresponding element definition is also made.

1. Class-Level Mappings
2. **Classes:** <packagedElement xmi:type=”uml:class”> corresponds to the generation of several things:
   1. A global element with the same name as the XMI name attribute, and of the complex type declared in the next step. If there is an attribute isAbstract with value true the global element will be defined as abstract.
   2. A complex type using a concatenation of the name attribute and the string “Type”. Within its complexContent this will contain at first an extension declaration with the name of the superclass, derived from <generalization>, via the xmi:id attribute, as base and then a sequence with an element definition for each <ownedAttribute xmi:type=”uml:Property”> following the next step. If no <generalization>is available, the base will be ddic:DDIObjectType
   3. If the class defines one or more properties and/or associations of the same name, as one of its superclasses, a complex type is generated, for which the name is a concatenation of the name attribute and the string “RestrictionType”. This type definition will contain all properties and associations from all superclasses, except those overridden in the same manner, as mentioned above but within a restriction to the superclass and not in extension. In this case, the extension base of the step above will be this restriction type and not the superclass.
3. **Associations to other DDI Classes:** If a child <ownedAttribute xmi:type=”uml:Property”> has an aggregation or association attribute, then resolve the ID in the association attribute, and take the value of the name attribute from the resolved XMI element. Concatenate the value of the name with the string “Reference”, and place an element declaration of type “Reference” there. The DDI referencing elements will always take their minOccurs and maxOccurs values from the value attribute of the <lowerValue> and <upperValue> XML elements (a value of “-1” means the maxOccurs has a value “unbounded”.
4. **Properties:** If a child <ownedAttribute xmi:type=”uml:Property”> has no aggregation and association attribute the type for the generated element has to be determined whether it is part of extended primitive definitions in DDI Core or Utility by the name attribute or one of the primitives already defined in XML schema, using the <type xmi:type="???"/> child of ownedAttribute. Cardinalities are handled the same way as for associations. The only exceptions from this are properties to be identified as language definitions (<type xmi:type="xs:language"/>). Those properties will become attributes to the defining complex type of the type xml:lang.
5. **Classes in the package “ComplexDataTypes”:** All properties (except a property named “content”) will become attributes to a complexType with simpleContent of the same type as the property “content”, if this class fulfills the following characteristics:
   1. It and its super classes include no associations.
   2. It or its superclasses have a property named “content”, that is not of type “anyURI” or “xhtml:BlkNoForm.mix”.

Naming conventions and name transformations

* In Drupal, all class name are written in upper camel case. All properties and associations are written in lower camel case.
* Within the transformation to XSD, all name of properties and associations are transformed to upper camel case, to fulfill the conventions of XML. In some cases, underscores and colons are erased, if they appear.