ODD chaining for Beginners

Lou Burnard *2016-10-22*

1 What is this for?

This little guide is intended to explain the mechanism of *ODD chaining*. An ODD file specifies a particular view of the TEI, by selecting particular elements, attributes, etc. from the whole of the TEI. But you can also refine such a specification further, making your ODD derive from another one. In principle you can chain together ODDs in this way as much as you like. You can use this feature in several different ways:

- you can add additional restrictions to an existing ODD, for example by changing the value list of an attribute
- you can further reduce the subset of elements provided by an existing ODD
- you can add new elements or modules to an existing ODD

2 How does it work?

An ODD can of course contain nothing but free standing declarations, using elements such as `<elementSpec>`, `<classSpec>` alone. But most TEI ODDs are made by reference to the huge existing collection of such declarations provided by the TEI Guidelines. An ODD such as TEI Lite or TEI Bare is composed of *references* to the objects it uses, expressed by means of elements such as `<moduleRef>`, `<elementRef>`, or `<classRef>`. These references (and also any additional free standing declarations) are collected together within a `<schemaSpec>` element which specifies the schema the ODD is intended to generate. This element has a useful but little known attribute *@source* the purpose of which is to state where exactly the objects referenced by the schema specification (the free standing declarations) are to be found. By default, when an ODD specifies no source, it is assumed that they are to be collected from the most recent release of the TEI Guidelines. You can modify this behaviour by supplying a different URI. For example, a `<schemaSpec>` with its attribute*@source* set to 'tei:2.4.0' would search for declarations in release 2.4.0 of the Guidelines. One with the value 'mySuperODD.subset.xml' will go looking for declarations in a file of that name in the current source tree. And one with the value 'http://example.com/superODDs/anotherSubset.xml' will go looking for it at the URL indicated.

It's important to understand that the resource indicated by the *@source*s attribute must contain complete and explicit specification elements: `<elementSpec>` rather than `<elementRef>`, `<classSpec>` rather than `<classRef>` and so on. It may of course contain other TEI elements, but these will be ignored entirely in the construction of a schema. A file called p5subset.xml, provided as part of every TEI release, is an example of such a resource: it contains specifications for every single TEI element, class, macro, and datatype, but nothing else much. If the *@source* parameter is not supplied, the most recently available version of this file is what will be used during the processing of an ODD.

3 Processing an ODD

Let's look more closely at the way the TEI defines a very light weight schema called TEI Bare. Its schema specification element begins like this :

```
<schemaSpec ident="tei_bare" xml:lang="en">
 <moduleRef key="core"
  include="p list item label head author title"/>
 <moduleRef key="tei"/>
 <moduleRef key="header"
  include="teiHeader fileDesc titleStmt publicationStmt sourceDesc"/>
 <moduleRef key="textstructure"
  include="TEI text body div front back"/>
 <classSpec ident="att.global"
  mode="change" module="tei" type="atts">
  <attList>
   <attDef ident="xml:space" mode="delete"/>
   <attDef ident="rend" mode="delete"/>
   <attDef ident="xml:base" mode="delete"/>
  </attList>
 </classSpec>
 <classSpec ident="att.fragmentable"
  mode="delete" module="tei" type="atts"/>
</schemaSpec>
```

No *@source* is specified, so declarations for the elements requested here will be taken from the current p5subset.xml.

Note that this ODD contains both references and specifications: there are references to modules (which may be thought of as short hand for references to elements or classes, since a module is simply a collection of element and class specifications) and specifications for two classes (`<classSpec>`), rather than references (`<classRef>`). The reference to the module tei brings with it specifications for most TEI classes, including these two. An ODD processor will therefore have to resolve duplicate class specifications for the classes att.global and att.fragmentable. The required resolution is indicated by the value of the *@mode* attribute: if this is 'delete' then both declarations are to be ignored, and the class is therefore suppressed; if it is 'change' then the two declarations are to be merged, with any part of it present in the second specification over-riding that in the first. In this case, the effect will be to suppress the three attributes mentioned.

If you'd like to check that this ODD does what you expect, and you have oXygen installed with a recent version of the TEI Frameworks, just download the file tei_bare.odd, and tell oXygen to apply the predefined transformation TEI ODD to HTML to it. This will produce a mini-manual for the TEI Bare customization in HTML format, near the beginning of which you should see a list of the elements the schema contains.

You may like to check that the modifications to the attribute class att.global indicated above have indeed been performed, by looking at its documentation in this mini-manual.

4 Rolling your own subset

In the preceding step, we processed the ODD with reference to the default p5subset, i.e with respect to the whole of the TEI. Suppose, however, that we would like to use TEI Bare as the starting point for another customization. We could simply edit the source of TEI Bare, and add our further modifications there, but that would soon become unmanageable if we were dealing with a larger customization as starting point. Instead, we will use TEI Bare itself as our source. To do this, as noted above, we need to generate a 'compiled' version of TEI Bare containing only specification elements in which all the references have been resolved. This is easily done using the stylesheet odd2odd.xsl which is supplied as part of the TEI Stylesheet package, but is not currently included in the TEI oXygen framework. There is however a command line utility teitoodd which does the job, and it is also easy to set up your own oXygen transformation to do it. We leave this as an exercise for the reader.

5 Chaining : subsetting

Suppose we now have a compiled version of TEI_bare in the file TEI_bare.compiled.xml. Processing the following schema specification should produce exactly the same results as we received from the uncompiled version.

```
<schemaSpec ident="Bare-prime"
 source="tei_bare.compiled.xml" start="TEI">
 <moduleRef key="tei"/>
 <moduleRef key="header"/>
 <moduleRef key="core"/>
 <moduleRef key="textstructure"/>
</schemaSpec>
```

This works because each of the <moduleRef> elements here refers to the module (i.e. set of elements etc.) *available in the compiled ODD* rather than to the module as defined in the whole TEI. Note also that simply supplying the compiled ODD as source for a schema is not enough : we must also specify which of the declarations it contains we want to use: *nihil ex nihilo fit...*!

However, the reason we started down this path was not to find yet another way of doing the same thing. Let's now make a reduced version of TEI Bare in which the <head> element is missing.

```
<schemaSpec ident="Bare-minus"
 source="tei_bare.compiled.odd" start="TEI">
 <moduleRef key="tei"/>
 <moduleRef key="header"/>
 <moduleRef key="core" except="head"/>
 <moduleRef key="textstructure"/>
</schemaSpec>
```

And, just for completeness, here is another way of achieving the same effect:

```
<schemaSpec ident="Bare-minus"
 source="tei_bare.compiled.odd" start="TEI">
 <moduleRef key="tei"/>
 <moduleRef key="header"/>
 <moduleRef key="core"/>
 <moduleRef key="textstructure"/>
 <elementSpec module="core" ident="head"
  mode="delete"/>
</schemaSpec>
```

Note that we cannot suppress or modify anything which is not already present in the compiled ODD specified by the *@source* attribute. This approach to ODD chaining works best in a situation where we first define an ODD which combines all the elements (etc.) that we ever plan to use, and then derive individual subset schemas from them, for example, one for manuscripts, one for early modern print, one for modern print etc. (This approach has been adopted by, for example, the Deutsches Text Archive.)

6 Chaining : supersetting

But ODD chaining is not restricted to subsetting. Suppose we want to take the pre-existing TEI Bare schema and *add* declarations from some other module. We could of course laboriously copy all the declarations we want into our <schemaSpec>, but it would be much nicer not to have to do that. Suppose for example that we want to add everything provided by the gaiji

module. That module was not included when we defined our compiled version of TEI Bare, though it is of course available in the full TEI. Here's one way of doing it:

```
<schemaSpec ident="Bare-plus"
 source="tei_bare.compiled.odd" start="TEI">
 <moduleRef key="tei"/>
 <moduleRef key="header"/>
 <moduleRef key="gaiji"
  source="http://www.tei-c.org/release/xml/tei/odd/p5subset.xml"/>
 <moduleRef key="textstructure"/>
</schemaSpec>
```

The <moduleRef> which pulls in the gaiji module uses its own *@source* attribute to specify where to find the declarations for that module. No sense looking for them in tei_bare.compiled.odd: they're not there. Instead, we will collect them from the online copy of the compiled ODD which provides the whole TEI Guidelines, as noted above. Of course we can also do the usual kind of subsetting : for example

```
<schemaSpec ident="Bare-plus"
 source="tei_bare.compiled.odd" start="TEI">
 <moduleRef key="tei"/>
 <moduleRef key="header"/>
 <moduleRef key="gaiji"
  source="http://www.tei-c.org/release/xml/tei/odd/p5subset.xml" include="g char
glyph"/>
 <moduleRef key="textstructure"/>
</schemaSpec>
```

This ODD will give us everything in tei_bare along with just g, char, and glyph from the default gaiji module. We could achieve the same effect by explicitly naming the elements we want, and again specifying where they are to be found:

```
<schemaSpec ident="Bare-plus"
 source="tei_bare.compiled.odd" start="TEI">
 <moduleRef key="tei"/>
 <moduleRef key="header"/>
 <elementRef key="g"
  source="http://www.tei-c.org/release/xml/tei/odd/p5subset.xml"/>
 <elementRef key="char"
  source="http://www.tei-c.org/release/xml/tei/odd/p5subset.xml"/>
 <elementRef key="glyph"
  source="http://www.tei-c.org/release/xml/tei/odd/p5subset.xml"/>
 <moduleRef key="textstructure"/>
</schemaSpec>
```

We can use this technique to put back an element which was deleted from the compiled schema. For example, the <q> element is not available in TEI Bare, but we can easily get it back. We can even specify which version of the <q> element we want: in this case, we will get the version defined in TEI P5 release 3.0.0 :

```
<schemaSpec ident="Bare-plus"
 source="tei_bare.compiled.odd" start="TEI">
 <moduleRef key="tei"/>
 <moduleRef key="header"/>
 <elementRef key="q" source="tei:3.0.0"/>
```

```
  <moduleRef key="textstructure"/>
</schemaSpec>
```