How to Make an ODD Automagically

Lou Burnard

2017-01-03

We keep telling you to make an ODD for your project. Surely there must be a way of kickstarting the process automatically? Of course there is. This Guide shows you how to generate an ODD automatically from a corpus of existing TEI P5 documents, and makes some suggestions about how you might want to improve it.

1 Setting up the ODD by Example Stylesheet

The TEI provides a utility called ODD By Example which will process a set of existing TEI documents and extract from them a description of the TEI elements and classes they use. This utility is an XSLT stylesheet, forming part of the standard TEI Stylesheet package. You will find it in the file tools/oddbyexample.xsl, within whatever folder you have used to install the Stylesheet package; on a typical Linux system, this will be called /usr/share/xml/tei/stylesheet. We don't describe here how to install the Stylesheet package.

It is a little different from the typical XSLT stylesheet in that it is designed to process a large number of separate documents rather than a single one. You control its input and output by means of parameters which must be passed to the XSLT processor in slightly differing ways depending on the processing environment. For example, you could run the stylesheet at the command line, using a command such as saxon directly, or you could run it within the oXygen framework, having first set up an appropriate *transformation scenario*.

If you are used to working at the command line, this may be the quickest and simplest option. To process all the TEI files in a directory called /home/me/myTEICorpus and produce a sample ODD file called myGenerated.odd in the current directory, you would issue a command like the following: `saxon /usr/share/xml/tei/stylesheet/tools/oddbyexample.xsl -o:myGenerated.odd -it:main corpus=/home/me/myCorpus`

(In case you are wondering, the `-it` option tells saxon which template in the stylesheet should be processed first.)

To define an appropriate oXygen transformation scenario for the same files, you would proceed as follows:

- In oXygen, open any TEI XML file in the /home/me/myTEICorpus folder
- Choose Transformation -> Configure Transformation Scenario(s) from theDocument menu
- Click New and choose "XML Transformation with XSLT"
- Give your scenario a name ("oddGenerator" for example)
- Leave XML URL as it is. Change XSL URL to point to the stylesheet oddbyexample.xsl in your TEI Framework directory directory. Enter `${frameworks}/tei/xml/tei/tools/oddbyexample.xsl` to find it
- Choose Saxon-PE 9.4.0.4 as processor
- Click the little yellow wheel next to this window to select Advanced Options: you need to set `Template("-it")` to `main`
- Click the Parameters button : you need to set the `corpus` parameter to contain the full name of the folder which you want to analyse. Assuming you opened one of its files in the first step above, just set the parameter to `${cfd}` and click OK
- Now select the Output tab ...

  - In the Save as window supply an output filename such as myGenerated.odd
  - Tick the Open in editor box
  - Select the XML radio button underneath Show in results view as and click OK

- Launch the transformation by clicking the Apply Associated button

Once you have defined this transformation scenario, you can use it as often as you like with any collection of files. You don't need to go through the whole of the above rigmarole every time! Next time round, proceed as follows:

- In oXygen, open any TEI XML file in the collection you want to process
- Choose Transformation -> Configure Transformation Scenario(s) from theDocument menu as before
- You should see the Scenario you just defined in the list of "Global" scenarios. Check the box next to it and click the Apply Associated button

Once this association has been made, every time you open that TEI XML file in oXygen, you can rerun the transformation, just by clicking the big red triangle on the tool bar (or typing CTRL-SHIFT-T, or selecting Document - Transformation - Apply Transformation Scenario)

You can also edit the scenario, for example by changing the value of the parameters passed on to the stylesheet, or by changing the output options. See below for a list of the parameters you can modify.

2 Understanding the output

The ODD generated by oddbyexample looks a bit strange at first. Scan the generated ODD for `<moduleRef>` elements, since these are particularly useful. You should see lines like the following

```
<moduleRef key="namesdates"
 include="persName listPerson person"/>
```

This tells me that my corpus uses the elements `<persName>`, `<listPerson>` and `<person>` which are supplied by the TEI module namesdates. No other element from that module is used in my corpus.

However, my ODD also contains many lines which are less useful. For example, if one of the global attributes (such as @corresp) has been used on just one of the elements in your corpus, the generated ODD has to delete it explicitly from every element on which it is *not* been used. More annoyingly, perhaps, the generated ODD always contains declarations of all module-specific classes whether or not elements from that module have actually been used. The declarations have no effect on the generated schema, but they make it harder to understand what the ODD itself is doing.

More usefully, the generated ODD can tell you about the values which are actually used for attributes (such as *@type*) which are of datatype teidata.enumerated.

Here's an example of the sort of thing I mean:

```
<elementSpec ident="persName" mode="change">
 <attList>
  <attDef ident="corresp" mode="delete"/>
  <attDef ident="when" mode="delete"/>
  <attDef ident="notBefore" mode="delete"/>
  <attDef ident="notAfter" mode="delete"/>
  <attDef ident="key" mode="delete"/>
  <attDef ident="type" mode="change">
   <valList mode="add" type="closed">
    <valItem ident="actor"/>
   </valList>
  </attDef>
  <attDef ident="subtype" mode="delete"/>
 </attList>
</elementSpec>
```

This specification tells me that in my corpus the only value actually specified for the attribute *@type* on the element `<persName>` is 'actor'. Other instances of this element might not specify a *@type* at all, of course, but this is the only value actually used. It also tells me that most of the other attributes made available by classes of which `<persName>` is a member are never actually used. For example, the attributes *@when*, *@notBefore* and *@notAfter* are all supplied by the class att.datable, of which `<persName>` is a member. These attributes are not used on any instance of `<persName>` in my corpus. They are however used on instances of at least one other member of that class (`<date>` in fact) and so the attribute class itself cannot be deleted from the ODD. Instead, as we see here, they must be explicitly deleted from each element which does not use them.

3 Using and improving a generated ODD

What can we do with this ODD file, other than study it as a witness to the follies of our encoders? We can of course process it to generate a schema and a mini-manual documenting our practice, just like any other ODD. Here's a reminder of how to set up an oXygen transformation scenario to do just that:

- Open your generated ODD file in oXygen
- In the Document menu, choose Transformation -> Configure Transformation Scenario(s), or type CTRL-SHIFT-C, or click the spanner icon on the toolbar.
- A list of available transformation scenarios is displayed. Tick the box next to those you want to apply: probably TEI ODD XHTML and TEI ODD to RELAXNG Compact
- Click the Apply Associated button to start the transformation(s); you can safely ignore the messages that scroll by in the lower window
- All being well, XHTML will be displayed using your default browser, while the RELAXNG Compact output will be saved in a folder called out, which oXygen will create in the current directory if necessary.

You should of course check that the schema generated from this ODD does in fact validate all your corpus files correctly though it would be somewhat alarming if it did not. When you have some new files to add to your corpus, however, this process becomes very useful, whether you decide to maintain the encoding practices already established, or to expand them to cater for new usages in your new files. Maybe you'll need to add new values to the permitted range for one of your attributes? Maybe an attribute or element you thought you would never use needs to be restored to the ODD?

Suppose for example that although your ODD says that the values of *@type* if supplied on `<div>` should be 'chapter' or 'section', your new material includes some `<div>` elements with a *@type* of 'book'. You might decide that this is a mistake and change the data, or you might decide that you need to modify the ODD. (Michael always used to say 'If there is a discrepancy between the text and the schema, in our world we trust the text') Let's say you adopt the latter course. You need to locate the `<valList>` within the `<attDef>` concerned, and add a new `<valItem>`. So where your ODD perhaps looks like this:

```
<elementSpec ident="div" mode="change">
 <attList>
  <attDef ident="type" mode="change">
   <valList mode="add" type="closed">
    <valItem ident="chapter"/>
    <valItem ident="section"/>
   </valList>
  </attDef>
 </attList>
</elementSpec>
```

You need to change it to read

```
<elementSpec ident="div" mode="change">
 <attList>
  <attDef ident="type" mode="change">
   <valList mode="add" type="closed">
    <valItem ident="chapter"/>
    <valItem ident="section"/>
    <valItem ident="book"/>
   </valList>
  </attDef>
 </attList>
</elementSpec>
```

While you're there, you might like to document what you mean by each of these values, for the benefit of non-English speakers, or yourself when you have forgotten whether 'section's contain 'book's or vice versa. You can do this by adding one or more `<desc>` elements within the appropriate `<valItem>`, like this

```
<elementSpec ident="div" mode="change">
 <attList>
  <attDef ident="type" mode="change">
   <valList mode="add" type="closed">
    <valItem ident="chapter">
     <desc xml:lang="en">the smallest subdivisions
           of a text</desc>
     <desc xml:lang="fr">les divisions les plus petites d'une
           texte</desc>
    </valItem>
    <valItem ident="section">
     <desc xml:lang="en">a group of
     <soCalled>chapter</soCalled>s within a
     <soCalled>book</soCalled>
     </desc>
     <desc xml:lang="fr">un regroupement de
     <soCalled>chapter</soCalled>s au sein d'un
     <soCalled>book</soCalled>
     </desc>
    </valItem>
    <valItem ident="book">
     <desc xml:lang="en">the largest subdivisions of
           a text</desc>
     <desc xml:lang="fr">les divisions les plus grandes d'une
           texte</desc>
    </valItem>
   </valList>
  </attDef>
 </attList>
</elementSpec>
```

Once you've made these changes, you might like to regenerate a schema from your ODD, and then check that oXygen will use the new features you added. When adding a `<div>` to a new document, for example, you should be offered a menu for the *@type* attribute including the new values and their associated help text.

Now take a look at the HTML 'mini-manual' generated from your ODD. By default this will contain just an index of links pointing to detailed specifications for each of the elements and classes your corpus uses. These descriptions are modelled closely in appearance on the detailed specifications used in the Guidelines proper.

By default, the examples, descriptions, and cross references supplied for each element will be just the same as those in the Guidelines. The content model and list of attributes however should reflect any changes proposed by your ODD.

You might like to test this by replacing one or more of the usage examples provided for each element with examples taken from your own corpus.

Open your generated ODD in oXygen and locate the `<elementSpec>` for the element you wish to change, if there is one. (If there isn't, you will need to add one yourself, of course.) For example, suppose I want to supply a better example for the `<hi>` element which has been included in my ODD. A declaration has been provided for this element deleting two global attributes which it does not use :

```
<elementSpec ident="hi" mode="change">
 <attList>
  <attDef ident="facs" mode="delete"/>
  <attDef ident="resp" mode="delete"/>
 </attList>
</elementSpec>
```

My new usage example is supplied following the `<attList>`. It is encoded using the `<exemplum>` element, which contains an `<egXML>` element and (optionally) some additional paragraphs of discussion.

```
<elementSpec ident="hi" mode="change">
 <attList>
  <attDef ident="facs" mode="delete"/>
  <attDef ident="resp" mode="delete"/>
 </attList>
 <exemplum> <egXML xmlns="http://www.tei-c.org/ns/Examples"> Here's how I use
    it <hi>my example</hi> </egXML> </exemplum>
</elementSpec>
```

Further details about the `<exemplum>` element and its content are provided in the appropriate section of the Guidelines, of course. Note here that when used like this in a customization ODD, it replaces completely any existing `<exemplum>` element in the TEI source. You may like to modify your ODD like this, and regenerate the HTML minimanual to check.

Before doing so, however, we suggest you also add some discussion explaining what your ODD is for, documenting its components and intended usage. This informal documentation can be as simple or complex as you want, and you can (of course) use all the range of elements provided by the TEI to express it. Add at least a `<div>` or two, with some `<head>` elements, some `<p>`s and some `<list>`s... You can also use some special tag documentation elements provided by the tagdocs module, which enable you to distinguish element and attribute names in the markup, to embed formal descriptions of specific elements in the prose, and many other things. Make sure that your ODD document is validated against a schema which includes these specialised elements however: the schema tei_odds in the exemplars directory is intended for this purpose.

For example, you might expand your ODD to look like this:

```
<body>
 <div>
  <head>A minimal TEI schema for archival transcription</head>
  <p>This schema proposes a minimal subset of TEI elements, adequate to basic
     transcription of archival sources. </p>
  <p>Each document is transcribed as a separate <gi>text</gi> element with its
     own <gi>teiHeader</gi>. Subsections of the document are marked up using
     the <gi>div</gi> element. </p>
  <p>The following TEI elements are used to represent these components: <specList>
    <specDesc key="TEI"/>
    <specDesc key="header"/>
    <specDesc key="text"/>
   </specList>
  </p>
<!-- lots more prose and discussion and examples here-->
 </div>
 <div>
  <head>Formal Specification</head>
  <schemaSpec ident="minimalSchema"
   start="TEI">
<!-- declarations from your generated ODD here -->
  </schemaSpec>
 </div>
</body>
```

When an ODD like this is processed, the HTML (or other document formats) generated from it by an ODD processor will contain all the text you see here. The element `<specDesc>` will however be replaced by the content of the `<desc>` element within the specification of the element indicated by its *@key* attribute. The `<schemaSpec>` will also be processed, as you have already seen, to produce formal documentation. Give it a try!

There are many other things you might do once you start editing your ODD. This tutorial just suggests a few that are likely to be particularly useful in the process of defining a useful schema for an existing set of TEI documents.

4 Configurable parameters for the oddbyexample stylesheet

Not all of them work... see `https://github.com/TEIC/Stylesheets/issues/212`

Parameter	Function	Default
corpus	path to directory containing input files	./
schema	name for generated schema	oddbyexample
corpusList	process only these input files	(no constraint)
prefix	process only files with names starting like this	(no constraint)
suffix	process only files with extensions like this	xml
defaultSource	path to reference TEI declarations	http://www.tei-c.org/Vault/P5/current/xml/tei/c

keepGlobals	whether to enumerate all global attributes	false
attributeList	specific attributes whose values should be enumerated	(none)
enumerateRend	whether to enumerate values used for rendition	false
enumerateType	whether to enumerate values used for type	false
includeHeader	whether to enumerate attributes of elements in TEI headers	true
processNonTEI	whether to process non-TEI elements	false
debug	whether to print debugging information	false
verbose	whether to produce verbose messages	false