<oo>→<vil> XML and TEI for Slavic p...

Maintained by: David J. Birnbaum (djbpitt@gmail.com)
Last modified: Wednesday, 24-Aug-2016 07:26:01 UTC

Manuscript transcription exercise 1

1. Introduction

"Vita Pauli simplicis", 19 March from the *Codex Suprasliensis*

The *Codex Suprasliensis* is available on line (photographic facsimile, diplomatic transcription, and interlinear parallel Greek text) at http://suprasliensis.obdurodon.org. Select the link to the beginning of the "Vita of Paul the Simple" and take a look at the text.

2. Document analysis

As discussed in the first (XML) session of our workshop, the first step in creating an XML document, before you type any angle brackets, is document analysis. Look over the images and transcription and plan the project from at least two perspectives: the large, overall structure (e.g., folios and lines) and the small, in-line idiosyncrasies (e.g., corrections, insertions, personal names, etc.).

The TEI is designed to provide resources from which you, the developer, can select those that are appropriate to your materials and your research goals, so you are not expected to tag everything that is taggable. With that said, projects are likely to agree on the larger structural markup, which tends to be useful for everyone, and they are more often individuated at a lower level. For example, if you care about person or place names, but not about part of speech or morphological properties, you can tag for the former and leave the latter untagged (or vice versa, of course). For the purpose of this group exercise we'll provide suggestions about what to tag and what to ignore, but in Real Life you would make those decisions yourselves according to your research goals.

3. Getting started

Here's how to begin tagging a file in <oXygen/>:

1. Open <oXygen/> and create a new XML document. There are two ways to do that:
    - Click on the icon shaped like a piece of paper with a dog-eared corner on the far left edge of the menu bar. Type "TEI" where it says "Type filter text" to reduce your options, and then click on "All" under "TEI P5"and push the "Create" button.

      Note: In Real Life you wouldn't use TEI All, which allows all 600+ TEI elements, because that would let you accidentally insert elements that shouldn't appear in your specific document. What you would do instead is create a schema that allows only a customized subset of the available elements. For this exercise, though, we'll allow all possible elements.
    - Type Ctrl+n (for "new") instead of clicking on the icon and then follow the same steps as above.

   This creates a skeletal TEI document.

2. Inside the skeletal TEI document you've created in <oXygen/>, find the `<p>` element inside the `<body>`. Select and delete the contents of that element ("Some text here."), leaving only the start and end tags. The phrase "Some text here." is a placeholder in the skeleton, and you're going to replace it with the text you want to edit.
3. Inside the `<p>` start tag, erase the letter "p" and replace it with "ab". `<p>` is for paragraphs and `<ab>` is for "anonymous blocks", that is, chunks of text that are not semantically paragraphs or anything else that is easily defined. We'll use this catch-all element to contain the text of the vita. Enter a few blank lines between the `<ab>` start and end tags, so that you'll have a place to paste the text you're going to edit.
4. There are at least two ways to insert the text you want to tag inside your new TEI document at the proper place. Choose one and use it to insert the text of the Vita between the `<ab>` tags:
    - Click on http://vilnius.obdurodon.org/suprasliensis.txt to open it in your browser. Don't worry if you see funny characters or boxes; the characters are correct in the file, but your browser may not be using a font that renders them properly. Click inside your browser window and then type Ctrl+a to select all text, followed by Ctrl+c to copy the selection. This copies the text to your clipboard, an invisible buffer that lets you then paste it elsewhere. Now click on a blank line between the `<ab>` tags to position the cursor and type Ctrl+v to paste whatever is in the clipboard (the plain text transcription) into <oXygen>, so that it will be ready for markup.
    - Instead of clicking on http://vilnius.obdurodon.org/suprasliensis.txt to open it in the browser, right-click on it and download it to somewhere in your local file system, such as your desktop or a downloads directory. Then put the cursor between the `<ab>` tags in the TEI document and from the <oXygen/> menu select "Document", then "File", and then "Insert File …". Navigate to the file you downloaded, select it, and click "Open" and the file will be inserted at the cursor position.

4. The TEI header

Before we edit the text of the vita, let's prepare the TEI header. Every TEI document must have a header that contains metadata about the document, and the skeletal TEI document created by <oXygen/> inserts placeholders for a simple header. For more information about the TEI header, see Chapter 2 of the TEI Guidelines. The `<fileDesc>` element inside the TEI header may contain additional elements, but the three elements that <oXygen/> creates for you are the only ones that are always required. Here are the steps to populate those placeholders with real data:

1. The `<titleStmt>` element contains information about the title of your document, and inside the `<title>` tags you should now type a meaningful descriptive title. This is the title of the digital document you're creating (not of the source from which you copied the data), so although it could be something like "Vita of Paul the Simple", it could also be "Digital edition of the Vita of Paul the Simple", or whatever else you want. For more information about the `<titleStmt>` see Section 2.2.1 of the TEI Guidelines.
2. The `<publicationStmt>` may contain either a simple paragraph (as it does here) or richer, more complex structured data, with more elements. Replace the placeholder "Publication Information" text with information about the publication properties of your document. As with the title, this is publication information about the digital document you are creating, and not about the source. In this case you might write something like "Unpublished". For more information about the `<publicationStmt>` element see Section 2.2.4 of the TEI Guidelines.
3. The `<sourceDesc>` contains information about the source you used to create your electronic document. In this case you might write something like "http://suprasliensis.obdurodon.org" or, more elaborately, `<ref target="http://suprasliensis.obdurodon.org"/>Bulgarian Academy of`

`Sciences digital edition of the Old Church Slavonic <cite>Codex Suprasliensis</cite></ref>`. For more information about the `<sourceDesc>` element see Section 2.2.7 of the TEI guidelines.

5. Fonts

Optional: You may see boxes or other funny characters in <oXygen/> if you haven't installed a font that contains all of the characters used in the transcription. The correct characters are there whether you can see them or not, so you can just ignore the oddities and trust that everything will look correct when you eventually publish your text. If you find it difficult to work with your text without being able to read all of the characters, though, you can install a font with a character set that's richer than the defaults, and we use the free Quivira font for those purposes. If you'd like to use it, download it from the link above and install it into your computer in whatever way you usually use to install fonts. To tell <oXygen/> to use Quivira instead of the default system font, go to your <oXygen/> preferences and then choose "Appearance", then "Fonts", and then "Editor". Change the font to Quivira, and because Quivira is smaller than many other fonts, we recommend changing the size to 16 and checking the "Bold" box. Then click the OK button.

Remember that this step is optional. Your display may already look normal, and render all characters properly, with the default system fonts. And even if it doesn't, your data will contain the correct characters even if they aren't rendered properly. It is nonetheless difficult to do some of the editing we need if you can't read all of the characters properly, so if there are problems with your display, we'd recommend installing and using the Quivira font.

6. Document analysis

Before you tag your text, you need to conduct *document analysis*, that is, you need to identify the structure of the document and the features you'd like to tag. Here are a few thoughts:

- The first line of our plain text file is a supplied title, and not part of the text of the actual manuscript, so we'll want to move it out of the `<ab>` (which is for the transcribed text) and into an appropriate alternative element.
- Otherwise the plain text contains one line of text for each manuscript line, with separate, stand-alone lines that identify the locations of folio breaks. We'll want to encode information about the lineation, and we'll want to distinguish lines of text from the lines that only supply information about folio breaks.
- Within the lines, superscript letters have been wrapped in parentheses. These are *pseudo-markup*, and in our TEI edition we'll want to replace them with real markup, since there aren't any parenthesis characters in the manuscript. *No editorial punctuation (parentheses, brackets, vertical lines for line breaks, etc.) should ever be encoded as plain text.* Such information should always be encoded as markup. If you want to render superscript letters by surrounding them in parentheses, you can control that at the rendering stage of the editorial process, but at the editorial stage the character data should contain only characters that are present in the source. The only exception is that it is conventional to insert space characters to separate words even when the text is written continuously in the original. You may retain the continuous script if you want, but for most editorial and publishing purposes you'll find it more convenient to divide the words.
- Words that are continued across a line break are not marked, and although the manuscript doesn't distinguish when a line break corresponds to a word break and when it doesn't, we'll want to add that information so that we can later reconstruct the words of the text. Following the requirement never to insert editorial characters into the text, we'll use markup (rather than a hyphen) for this purpose.

- There are a few corrections in the text, which are marked in red or green in the online edition. For example, on line 3 of folio 86r an “и” has been corrected and replaced by “ѡ”.

  In the plain text file that you’re going to tag, we’ve included only the replacement text from each correction, with no mention of the text it replaces. In Real Life we’d be doing our own transcription and noticing the corrections in the process, but to save time for this exercise we’ve done the transcription for you, and we’ve transcribed only the replacement text, which means that you won’t see the replaced (deleted) text in the plain text file. You can see that text in the photographic facsimile (well, if you have very good vision!), and we’ll also tell you where the corrections are when it comes time to tag them for this exercise.

7. The <head> element

Since the first line is not part of the manuscript text, let’s move it out of the <ab>. Above the <ab> start tag, create a <head> element, which is the TEI element used to represent the title of a section (in this case, of the <body>). Select the first line of the plain text with the mouse, cut it (Ctrl-x), click between the start and end <head> tags, and paste the text (Ctrl-v). If the cut and paste operation left a blank line behind, delete it manually.

Note that it might seem natural to use <title> for this purpose, but the TEI has decided that the <title> tag has a different meaning: it is used for the title of a work, and not for the title of a section of a TEI document. You can read about the <head> element at Section 4.2.1 of the TEI guidelines and about the <title> element at Section 3.11.2.2. Having to look up which element to use for which purpose is tedious at first, but you’ll quickly memorize the elements you use most often. If you try to insert an element where it isn’t allowed (e.g., <title> is not allowed as the first child element of <body>), <oXygen/> will display an error message (try it!). But sometimes an element is valid in a particular position but has a meaning different from what you intend, and if in those cases you haven’t yet memorized the correct markup, you’ll have to look up the usage in the TEI guidelines.

8. Tagging for structure

8.1. Line breaks

The process of converting plain text to XML is called *up-conversion*. Whether we perform an up-conversion from the outside in or the inside out depends on our documents and our desired markup, but for this exercise we’ve decided to work from the outside in. For that reason we created the TEI superstructure first and pasted the text inside it. The biggest structural units are the lines, so we’ll start by tagging those, and we’ll then dig more deeply into the lines to tag superscription and substitutions or corrections.

The TEI markup for line breaks is to insert an empty <lb> element at the beginning of each line. To avoid having to type those individually, you can use a global find and replace operation. But because you want to insert this markup only inside the <ab>, you’ll want to restrict the search to only that context.

To perform a find and replace operation in <oXygen>, type Ctrl+f to open the find dialogue. Examine the interface for a moment. At the top you can enter the text you want to find (in this case, the beginning of a line) and the text you want to use to replace it (in this case, your empty <lb/> element tag). You can also restrict the search to just a particular location in the box labeled “XPath”, about which see below. You can ignore the “Direction” and “Scope” interfaces for now. Under “Options”, you should leave “Wrap around” checked; this ensures that the entire document will be searched no matter where the cursor is located (otherwise the search proceed only forward or backward from the cursor position and stops when it reaches the end or beginning of the text). You’ll want to check the box labeled “Regular expression”, about which see below.

In the “Find” box, type “^\s*” (without the quotation marks). *Regular expressions* are a system for representing textual patterns more complex than just strings of literal characters, and this regular expression has the following meaning:

- The inital caret (“^”) means “match the rest of this expression only at the beginning of a line”. Since we want to insert our `<lb>` elements at the beginning of each line, this ensures that we'll be working at the beginning of the line when we make our changes.
- The “\s” (pronounced “backslash-s”) matches any whitespace character, and the asterisk after it means zero or more instances of any whitespace characters. The beginning of your line of text may have space characters before it if <oXygen/> inserted them when you pasted it (<oXygen/> sometimes tries to align the text neatly for you), so in this case your pattern will match whenever there are zero or more whitespace characters immediately after the beginning of the line, that is, whether there is whitespace before the first real character of the line or not. This pattern, then, says “find the beginning of every line, along with any optional whitespace immediately after it, and replace it with whatever I type in the ‘Replace with’ input box”.

In the “Replace with” input box type “<lb/>” (without the quotation marks). Don't forget the trailing slash character, which is how you spell an *empty element* in XML, one that has just a single tag, instead of separate start and end tags. This says that whatever you match will be replaced by whatever you've typed here, which has the effect of inserting this text at the beginning of each line (while also removing any whitespace characters that might have been at the beginning of the line).

In the box labeled “XPath”, type “//ab” (without the quotation marks). We'll learn about XPath expressions later in the workshop, but as a preview, this item restricts the find and replace operation to text inside an `<ab>` element. This ensures that we don't insert `<lb>` elements inside our header or anywhere else where we don't want them.

By default, find and replace operations in <oXygen/> search for literal strings of text, and there aren't any literal carets or backslashes or Latin “s” characters or asterisks in our text. If you haven't done so already, now is the time to check the “Regular expression” box in the “Options” section. This tells <oXygen;> that what we're searching for is a regular expression (that is, a coded pattern), rather than a literal string.

Moment of truth: Before you do any replacement, push the `Find All` button on the right. The number of matches it finds will appear in the dialogue window, and the results should appear in a bottom window that <oXygen/> will open. If it doesn't look as if you've found anything, interrupt us and we'll troubleshoot the problem with you. Otherwise, if you're finding what you want to find, push the “Replace All” buttom to make the changes. If you mess up the replacement, you can undo whatever you did with Ctrl-z.

If you had whitespace before your closing `<ab/>` tag, the find and replace operation may have inserted an erroneous extra `<lb/>` just before it. If so, delete that manually.

Once you've finished your find and replace operations, you can close the dialogue window.

8.2. Folio breaks

The find and replace operation could not distinguish folio identifiers from real lines of text, and it therefore erroneously inserted `<lb/>` tags before the folio breaks. Fix those manually by removing the `<lb>` elements. Now change the text of those lines to TEI markup for page breaks, which is an empty `<pb>` element. It's conventional to represent the folio number by using the `@n` attribute, so that, for example, the line that marks the start of folio 87r would read `<pb n="87r"/>`. Whether you keep the leading zero is up to you; you won't want to render it when you later publish your edition, but you can either remove it now (during markup) or later (during conversion for rendering).

Fixing the folio breaks manually is not difficult in a small text like this, but you wouldn't want to have to do it manually in the full manuscript (270 folios). It is possible to write less crude regular expressions to distinguish lines that represent folio breaks from lines of real text, and that't what we did when we prepared the actual edition. Even if you aren't familiar with regular expression syntax, since you know that regular expressions match patterns, what sort of patterns might you search for in order to make that distinction?

9. Refining our structural markup

We tagged lines as part of our high-level structural markup, and we can refine that by dealing with word breaks at the ends of lines and with line numbering.

9.1. Word breaks

At the moment, line breaks that fall in the middle of words are not distinguished formally from those that fall between words. The TEI way of representing this difference is that line breaks between words are simply `<lb/>`, while those within words are `<lb break="no"/>`. Much as a human reader has to use linguistic knowledge to recognize when a line break divides a word and when it doesn't, you'll have to use your human knowledge to add the attribute name:value pair where needed. For the purpose of this exercise you needn't do them all, but do a reasonable number to get a feel for the process.

9.2. Line numbering

You don't have to number lines in your markup because XML processors can count. For example, the fifth `<lb>` element after the folio break for folio 86r represents the beginning of the fifth line of that folio. But because our text begins toward the bottom of folio 85v, automatic counting would not be able to determine the appropriate line numbers for those first three lines. We won't do anything with this now, but in Real Life we would add just enough line-numbering markup to provide the information that can't be deduced automatically, and if we wanted line numbers in our edition, we would let an XML processor (of the sort that we'll introduce later in the workshop) supply them.

10. Inline markup

The inline markup we'll work with for this exercise involves corrections and superscription.

10.1. Corrections

In this edition there are a few instances of corrections, such as the one at 86r3 mentioned above. The TEI way to represent this is with a `<subst>` (= "substitution") element, which contains a `<del>` for the text that was deleted and an `<add>` for the text that was added in its place. The `<del>` and `<add>` tags may appear in any order inside `<subst>`, but you'll make fewer errors if you train yourself always to use the same order. We put `<del>` before `<add>` because we think of correction as typically involving deleting something before writing its replacement, but that's just a human mnemonic, and the order of the child elements of `<subst>` is not informational.

In this edition we found two corrections; the other is at 86r18. You'll need to look at the photographs to see what the original text was, and once you've done that, mark them up as described above. The easiest way to tag existing text in <oXygen/> is to select the text you want to tag with the mouse, type Ctrl-e (for "element"), and then select the element you want from the drop-down list that will open in the dialogue. If

the plain text file you are tagging already contains the replacement text, as is the case in this exercise, we'd suggest tagging it first as `<subst>`. Notice that when you do that the text inside the new `<subst>` tags remains selected, so you can then tag it as `<add>` by leaving it selected and typing Ctrl-e again and selecting the `<add>` element. (Should you tag it first as `<add>`, you then have to select the text *and the new `<add>` tags* and wrap them in `<subst>`, which requires more work because you have to make two explicit selections.) Once you've introduced the `<subst>` and `<add>` tags, put the cursor between the `<subst>` and `<add>` start tags and create an empty `<del>` element, into which you can enter the deleted text. To create an empty pair of tags you can either use Ctrl-e without selecting any text, in which case the tags will be inserted at the cursor position, or you can type the start tag directly, in which case <oXygen/> will insert the end tag for you.

10.2. Superscription

In our plain text transcription we wrapped superscript letters in parentheses, but in an XML text we should use markup for that purpose. The TEI markup for superscription uses the `<hi>` element, which represents text that is highlighted in any way. It's conventional to treat superscription as highlighting (since it modifies the representation of the character), and to show that the highlighting in this case is specifically superscription we use the `@rend` attribute with the value "sup". For example, a superscript "ѫ" would be encoded as `<hi rend="sup">ѫ</hi>`. If there is a titlo or pokrytie or other diacritic over the superscript letter, that diacritic should also be included inside the `<hi>` tags.

Now try finding the superscript letters in the transcription and replacing the parentheses with markup. You can use the find and replace dialogue to find them; uncheck the "Regular expression" box (since now we're searching for a literal character), enter "(" (without the quotation marks) in the "Find" box, and hit "Find All". We found superscript characters at 86r22, 86v10, and 88v7.

Unicode does provide some separate superscript Cyrillic characters (see http://www.unicode.org/charts/, where the superscript characters are in Cyrillic Extended A and Cyrillic Extended B) but not all of them. We avoid using these in our own work for two reasons:

- Since the inventory of independent Unicode superscript Cyrillic characters is incomplete, in at least some cases we would have to use an alternative (markup-based) strategy. In order to avoid inconsistency in our documents, with a mixture of primary superscript characters in some cases and markup where primary characters aren't available, we recommend using only markup, since that can be done consistently.
- Unicode aims to distinguish as independent characters only those units of writing that have different semantics (and not only different shapes). It is not practical to maintain the distinction consistently because the semantics depend on the function (e.g., upper ~ lower case has different semantics for rendering but not normally for sorting or searching). Whether superscript characters should be regarded as fundamentally semantically different from their regular counterparts is unclear, but if they are understood as presentational variants of the regular characters, their appearance *should* be represented through markup, like a difference in font, and not through separate Unicode characters.

11. Tagging persons

The TEI provides a mechanism for tagging persons that can simplify document processing (such as structured searching or the creation of descriptive statistical reports) in situations where the same person may be referred to in different ways. If Paul were always called "Paul", we could just search for the string, but because he can have different declensional endings, and can also be identified by an epithet or a personal pronoun, there is no convenient string search that will find all references to Paul in the text. If the ability to find those types of references is part of the research goals of the project, one way to deal with this limitation is to front-load the work by identifying the different occurrences during markup, adding a consistent identifier that will enable the retrieval of all related forms later. The TEI provides mechanisms for supplying supplementary information about persons, so in addition to being able to

retrieve all references to Paul, this type of markup would make it easy to find all male persons, or all monks, or all persons with any properties that have been introduced into the markup.

If you don't care about being able to find persons, whether individually or according to shared properties, there is no need to add this type of markup. But if that sort of access is a requirement for your project, the TEI way to facilitate it is through a *personography* (list of persons and their properties), coupled with attributes within the body of the text that point to those lists. Here's how to create and implement a personography.

11.1. The TEI personography architecture

A TEI personography is implemented as a `<listPerson>` element—which is, perhaps not surprisingly, a list of persons. The personography is placed in a particular location in the header, and it contains biographical information about all persons in the document. Wherever those persons appear, they are then tagged with markup that points to the personography entry. In this way, an accusative or genitive singular form of Paul's name could be tagged along the lines of `<name ref="#paul">павъла</person>` to indicate that information about the referent of the pronoun in that location can be found by *dereferencing* (following and looking up) the pointer. That all happens during processing; what happens during markup is that we need to 1) create a personography and 2) tag the references to persons in the text.

11.2. The personography file

The personography is located inside the header and has the following structure:

```
<profileDesc>
 <particDesc>
  <listPerson type="historical">
   <person xml:id="paul">
    <persName>Paul the Simple</persName>
   </person>
   <person xml:id="anthony">
    <persName>Anthony</persName>
   </person>
    <!-- ... -->
  </listPerson>
 </particDesc>
</profileDesc>
```

The `@xml:id` attribute is the value to which all references to the person will point inside the main text. These values must be unique within the document; a document that has two identical `@xml:id` values is not a valid XML document. There are some restrictions on the values allowed for this attribute (e.g., no spaces are allowed; digits are permitted but not as the first character, and a few others), and the simplest strategy is to use only alphabetic characters. Because XML markup is case sensitive, "Paul" and "paul" are different values. It is common (but not required) to use lower-case letters for single words and camel case (e.g., "paulTheSimple") or underscores (e.g., "paul_the_simple") for values that consist logically of more than one word.

The personography can contain much more information than just the personal name and the `@xml:id`. If you'd like to enrich your personography, you can look up other possible content at Section 13 of the TEI guidelines.

The `<profileDesc>` goes inside the header after the `<fileDesc>`. The persons in this text are our hero Paul the Simple, his confessor Anthony, the man possessed by demons, and there may be others. Create a personography in your TEI file similar to the one above.

11.3. Pointing into the personography

The simplest way to associate persons in the main text with persons recording in the personography is by tagging names in the main text as `<name>` and adding a `@ref` (for "reference") attribute that points to the `@xml:id` value of the referent. The syntax for pointing is to precede the `@xml:id` value with a hash mark ("#"), so, for example, in 85v28–30 we might tag the reference to Paul as:

```
<lb/>Мѣсаца марта въ девѧтъı на десѧте· житие прѣ-
<lb/>подобьнаго ѻ҃тьца нашего· <name ref="#paul">павъла прѣ-
<lb/>простаго</name> ·⁖·
```

This presumes that we have an entry in our personography with an `@xml:id` value of "paul". Note that the `@xml:id` value *does not* have the hash mark, but the `@ref` that points to it does.

The `<name>` element should be used only with actual names. If you want to tag pronouns or epithets that point to particular persons, the TEI way to do this is with the `<rs>` ("referring string") element, which takes the same `@ref` attribute as `<name>`.

Spend a few minutes creating personography entries and then tagging persons (whether as names or pronouns) in the text.