EVT Frequently Asked Questions

** Work in progress **

EVT 1

Where is my image?

One of the most frequent request for help by EVT users, especially neophytes, concerns the loading of images in the viewer after the XSLT transformation is complete. Sometimes the text shows just fine, but the image frame is empty: why is that? In the 99% of cases, it is just a problem of file names, remember that

- the name of each image **must** be the same as the @xml:id value of the corresponding <pb/>, so if you have your page breaks starting at MS_folio_001 (that is, `<pb xml:id="MS_folio_001"/>`) then your first image must be named `MS_folio_001.jpg`:
  - `<pb xml:id="MS_folio_001"/>` ↔ `MS_folio_001.jpg`
  - `<pb xml:id="MS_folio_002"/>` ↔ `MS_folio_002.jpg`
  - `<pb xml:id="MS_folio_003"/>` ↔ `MS_folio_003.jpg`
  - …
- the extension doesn't count, but do not forget the _big and _small **suffixes** for high resolution images and thumbnails, respectively, so that you will have three versions of each image: `MS_folio_001.jpg`; `MS_folio_001_big.jpg`and `MS_folio_001_small.jpg`;
- the same is true for **double-side** images, but here you will have to be even more careful due to the naming convention used by EVT.

Is it possible to add another language to the UI? How?

Yes, it is possible to further extend the user interface localization adding new languages. You just need to prepare a JSON file for the new language and save it as **`new_language.js`**; remember that you will have to change "new_language" to the actual name of your language (e.g. `español.js`). Then, perform the following steps:

- put your **`new_language.js`** file in the **`config/langpack`** folder (be sure that the beginning of that file is **`jquery_lang_js.prototype.lang.new_language`**);
- open the file **`builder_pack/modules/html_build/evt_builder-callhtml.xsl`**;
- add **`<script type="text/javascript" src="{$html_path}/config/langpack/new_language.js"/>`** after the other language declarations;
- download an image of the flag for the new language and put it in the **`images`** folder: look up the dimension of the images of the other flags, the new image size should be the same or very similar;
- add **`<img src="images/new_language.gif" class="flag" data-value="new_language" lang="def" title="*new_language_LABEL-FOR-TITLE*" alt="*new_language_LABEL-FOR-ALT*"/>`** after the other, where **`new_language.gif`** is the name of the image for your flag;

- remember to add a string in every language file for **`"*new_language_LABEL-FOR-TITLE*": "New language"`**, so that every language will have the translation for the new flag title.
- rebuild the edition output: just follow the instructions in the *EVT Manual - section 4. Step by Step Instructions*;
- Reload the index in the browser and you will see the changes.

If you want to change the default language of the interface, just open the file js/plugin/jquery-lang.js and change the value of variable jquery_lang_js.prototype.currentLang to "new_language" (it should be at line 40).

How can I add a translation for my edited text?

As you probably have noticed, EVT for the moment only handles two levels of edition (Diplomatic and Interpretative), but is ready to be expanded in order to manage more levels and different versions of the same text, such as one or more translations, historical editions by previous editors, sources and analogues. The problem is that this is not a simple neither quick feature to develop, however it surely is in our scheduling of "task to do in the future". At the present moment, there are two possible ways to add the translation text: 1. consider it as if it were one more edition level, or 2. manually add the HTML files holding the translation to the EVT-produced edition.

The method to add a new edition level is as follows.

First you have to add a new <edition> element in the edition_array variable in evt_builder-conf.xsl:

```
<xsl:variable name="edition_array" as="element()*">
       <edition>Diplomatic</edition>
       <edition>Interpretative</edition>
       <edition>Translation</edition>
</xsl:variable>
```

Then you will have to add a new **`xsl:if`** condition in the **`edition_leve`**`l` template, and put your transformation code inside of it:

```
<xsl:template name="edition_level">
     <xsl:param name="pb_n"></xsl:param>
     <xsl:param name="edition_pos"/>
     <xsl:if test="$edition_pos=1"> [...] </xsl:if>
     <xsl:if test="$edition_pos=2"> [...] </xsl:if>
      <xsl:if test="$edition_pos=3"> * [...] * </xsl:if>
</xsl:template>
```

Then, you will have to create a set of other templates with the proper transformations you want to be displayed in the translation page, setting the `mode="translation"` (the same name used in the `<edition>` element, lower case).

The only problem with this method is that you need to have everything in one file as it is very difficult to access other XML files from this point of the transformation chain, unless you make use of **XInclude**, of course, which would let you keep the translation separated from the edition. Another disadvantage is that we are mixing two actually different things here, for the future translations and

other versions of the text (e.g. sources and analogues) will surely be handled as a separate case than edition levels, i.e. there will be a separate selector.

There is another way to do the trick, that is faster but less optimal and involves manual changes to the output data:

- modify the structure.xml file in the output folder adding a new <edition> element after the other;
- add the html files with the translation in the proper folder, that is /data/output/**translation/** (the folder name needs to be the same you used in the <edition> element, all lower case);
- the content you want to be displayed needs to be inserted in this structure, otherwise it won't work:

```
<div id="text_frame">
      <div id="text"> [your translation] </div>
</div>
```

- name every file in the proper way: page_*xml:id*_translation.html (the *xml:id* value refers to the @xml:id attribute of the relative <pb>);
- make a backup of these files because every time you will launch the XSLT transformation the entire output folder will be overwritten.

It is not optimal, since every new edition building will overwrite the data/output folder, but for the moment it is probably the faster way if you prefer not to add your own XSLT templates.