EVT 2.0 - Editor Manual

For EVT 2 beta2

Work in progress

1. Introduction

1.1 About EVT

EVT (Edition Visualization Technology) is a light-weight, open source tool specifically designed to create digital editions from texts encoded according to the TEI XML schemas and Guidelines, freeing the scholars from the burden of web programming and enabling the final users to browse, explore and study digital editions by means of a user-friendly interface.

This tool was born in the context of the Digital Vercelli Book project, in order to allow the creation of a digital edition (which has been available in beta form for more than two years) of the Vercelli Book, a parchment codex of the late tenth century, now preserved in the Archivio e Biblioteca Capitolare of Vercelli and regarded as one of the four most important manuscripts of the Anglo-Saxon period as it regards the transmission of poetic texts in the Old English language. However it has evolved into a tool suitable to fit different texts and needs. For example, it is now being used to publish the digital edition of the Codice Pelavicino manuscript, a medieval codex preserving charters dating back to the XIII century. The continuous development and need to adapt it to different types of documents and TEI-encoded texts has shifted the development focus towards the creation of a more general tool for the web publication of TEI-based digital editions, able to cater for multiple use cases.

The entire structure of the software has been remodeled, in order to make it lighter, more usable and more adaptable; we decided to use the Model View Controller (MVC) approach, that is a very common architectural pattern in object-oriented programming, that allows to separate the logical presentation of the data, from the application logic and the processing core. Wanting to maintain the original feature of EVT, and therefore do not give up the client only approach, we decided to use AngularJS, a JavaScript framework inspired by the MVC programming logic, especially suitable for the development of client-side Web applications; among other things, this framework allows to define custom HTML components and use the data-binding mechanism to associate the model of the data to the UI elements, and manage the updates of the latter avoiding the direct DOM manipulation.

1.2 How it works

Before the refactoring, EVT was composed of two main units: EVT Builder, for the transformation of the encoded text using special XSLT 2.0 templates, and EVT Viewer, for the visualization into a browser of the results of the transformations and the interaction with them. The idea under the new version of EVT is instead to leave to EVT Viewer the task of reading and parsing with JavaScript functions the encoded text, and “save” as much as possible within a data model, that persists in the client main memory, and is organized in a way that allows a very quick access to the data in case of need. This has obviously led to the elimination of the EVT Builder level, and therefore it allows to open a digital edition directly in the browser without any previous XSLT transformation.

Please note that starting from version 67 Firefox developers adopted the same security-conscious policy chosen by developers of Chrome and other Web browsers, that is forbidding loading local files (= documents available on the user's computer drive) in the browser as a result of the execution of Javascript programs. The goal is to improve global security when browsing the Web, but the unpleasant collateral effect is that of preventing the loading of digital editions based on EVT, or similar software, from local folders. Fortunately there are several workarounds that can be used to test EVT editions that are located on your hard drive:

- option no. 1: launch Chrome from the command line with the `--allow-file-access-from-files parameter`, after that you can press CTRL+O to open the `index.html` file, or you can just drag and drop it on Chrome's window; this is probably the most simple way to do it;
- option no. 2: download and install Firefox ESR v. 60: this version predates the new security policy adopted in FF v. 67 and, furthermore, it can be installed in parallel with any other version of Firefox;
- option no. 3: install an extension providing a local web server on Firefox or Chrome, f.i. there is this one available for Chrome.

This problem, however, only affects local testing, after the edition has been uploaded on a server there are no problems in accessing it with any of the major browsers.

1.3 Main features

At the present moment EVT can be used to create both diplomatic and critical editions with multiple levels of apparatuses, encoded using the TEI Parallel Segmentation Method. This means that e.g. a diplomatic transcription or a critical edition encoded according to the TEI *Guidelines* should already be compatible with EVT 2, or require only minor changes to be made compatible.

Among the main features you will find:

- **Critical edition support**. Enlarged critical apparatus, sources apparatus and analogues apparatus, variant heat map, witnesses collation and variant filtering are some of the main features developed for the critical edition support.
- **Bookmark**. Direct reference to the current view of the web application, considering view mode, current document, page and edition level, eventual collated witnesses and selected apparatus entry.

- **Named entities and lists of entities**. Each named entity can be highlighted in the text, opened to browse available information and searched in dedicated lists.
- **VisColl support**. The VisColl XSLT stylesheets have been integrated and adapted so that you can create and explore the quire structure of a manuscript.
- **3DHOP support**. An initial, experimental support for the 3DHOP software has been added to EVT.
- **Interactive bibliography**. Users can visualize the bibliography of the edition and reorder the entries by author, publisher or publishing date.
- **High level of customization**. The editor can customize both the user interface layout and the appearance of the graphical components.

2 A short guide to EVT

EVT 2 can be used to prepare an edition right away, immediately after installing it on your hard drive: see the *Installation and management of the edition data* section first, then *Configuration*, to understand how EVT works and how you can use it to publish your editions. A more detailed guide will be published separately, as a reference manual, and will also include instructions about customization.

If, on the other hand, you are interested in **developing** a specific functionality in EVT 2, or in modifying an existing one, we suggest that you download and install the *Development framework*. The `README.md` contained in it explains how to install and configure the development framework needed for this purpose. This step is only needed if you want to start working with EVT source code, so it is in no way necessary for basic users.

2.1 Installation and management of the edition data

Installation is quite simple, download the compressed archive from EVT home page, unzip it in a suitable location on your hard drive, and you are ready to use it with your edition files. Within the main folder there are only two folders which should be modified by the user:

- **`config`**: here you'll find the config.json file which is the main configuration file for EVT, but also custom-style.css where you can add your custom CSS rules for TEI elements visualisation (see below sections 2.2 and 2.3);
- **`data`**: here go all of your edition data, including the TEI-encoded documents, images, and other edition files.

Everything else should not be modified, unless you know what you are doing very well. It is in fact possible to modify the JavaScript parsers, but doing so directly in an EVT installation is less efficient than doing it on a GitHub repository: since EVT is an open source tool you are welcome to fork it on GitHub and change the existing parsers and/or add your own parsers.

Before moving to the configuration step (section 2.2) you should copy the different edition components in the `data` folder and sub-folders, this is how your files should be organized:

- your encoded edition document must be copied in the `data/text` directory, actually EVT comes with a default directory structure, distinguishing between images, text and other types of data:

`data/images`	put your images here, you will find some sub-folders (e.g. `data/images/single`, `data/images/hotspot` etc.), create more if needed
`data/models`	put your 3D models here, again you will find `multires` and `singleres` subfolders
`data/text`	put your textual data here, note that there are some subfolders already: `documents` `schema` `sources` and more may be added as needed (e.g. `witnesses`)
`data/viscoll`	put all VisColl-related files here

  You can modify this structure as desired, provided that the appropriate paths are specified in the configuration file (`config.json`, see below) and that all user data is added within the `data` folder;

- to have your edition parsed and loaded by EVT you have to point to it explicitly modifying the `config.json` file in the `config` directory and specifying the name of the main file:

  ```
  "dataUrl": "data/text/My_edition.xml",
  ```

  While this is the most important configuration option, since it tells EVT where to start with your edition, note that there are several other options available in that file, so that you can customize the layout and appearance of your edition (see section 2.2). Also note that some configuration options may be necessary to make desired features available, for instance to add the required edition level, so make sure you read the following section and check the default configuration file.

- optionally, you can add your own CSS instructions to modify the appearance of specific TEI elements by editing the `config-style.css` file in the `config` directory. See below section 2.3.

2.1.1 Images

EVT supports the most common image formats thanks to the embedded viewer, OpenSeaDragon. An experimental version with support for large images using the tiling method has been created successfully, but it will be integrated in the main development tree at a later moment.

The miniature images shown in the thumbnail view[1] are generated automatically from the main images used to show the manuscript folios besides the transcription or edited text: it is important to tell EVT where to find them, which is done by writing the correct path as a value for the `singleImagesUrl` configuration parameter, the default path is to `data/images/single` (see below section 2.2).

Synchronization of image and text at page level, for diplomatic editions, works thanks to the `@facs` attribute in page elements (`<pb/>`), see 3.3.2 for an example. To link text and image at line level, or to create hotspots leading to pop-up windows on the image itself will require a `<facsimile>` element holding `<surface>`s and `<zone>`s, see 3.3.1.

2.2 Configuration

There are several configuration options, ranging from the folders where edition data is stored to User Interface layout and available tools, that can be set by editing the `config.json` file in the `config` directory. Below you will find a detailed list of the available options: in the file you will see a list of options on the left, to configure EVT you will have to insert the appropriate values in the textual fields on the right. Sometimes those values will consist of boolean strings ("true" or "false"), sometimes they will be simple character strings (e.g. "Interpretative edition"), in other cases you will have to enter TEI XML elements (e.g. ", "); for colors it will be necessary to specify the correct RGB values (e.g. "rgb(108, 145, 207)").

If you find this file difficult to read and/or change you can try out the beta EVT2-Config-Generator: upload the current config.json, change the parameters you need to change and download the new `config.json`. Note that this is the first version of the EVT2 Config tool, so there may be glitches and/or problems (some options may be missing, f.i.), please report them to us.

Main edition data

Edition main information

- `indexTitle`. Choose a title for your edition. If you leave it blank the default "EVT Viewer" title will be shown.

[1] Accessible through the corresponding button in the navigation bar or in the image frame if the latter is not available.

- `webSite`. If you specify an external web site there will be a link pointing to it.
- `logoUrl`. You can also add a custom logo that will appear before the edition title: just indicate the path to it; it can be a URL or a relative path: we suggest that you put it into `data` and point to it (f.i. `data/icons/myLogo.jpg`).

Source files

- `dataUrl`. Indicate the file name of the XML file of your encoded edition. It can point either to an internal folder or to an external online resource.
- `sourcesUrl`. Indicate the file name of the XML file encoding the list of all the bibliographic references for the sources apparatus. It can point either to an internal folder or to an external online resource.
- `analoguesUrl`. Indicate the file name of the XML file encoding the list of all the bibliographic references for the analogues apparatus. It can point either to an internal folder or to an external online resource.
- `sourcesTextsUrl`. Indicate the folder where you intend to put the XML containing the texts of external sources (if you have any).
- `singleImagesUrl`. Indicate the folder where you intend to save the edition images, default path is to `data/images/single`.
- `enableXMLdownload`. Decide if you want to enable the XML download (`true`) or not (`false`).

VisColl files

- `visCollTextUrl`. Currently unused, please ignore.
- `visCollStyleUrl`. Currently unused, please ignore.
- `visCollSvg`. Indicate the folder where you intend to save the SVG images generated by the VisColl XSLT style-sheets, default path is to `data/viscoll/SVG/`. Note that you have to generate the images using an XSLT 2 processor, such as Saxon 9 (standalone product or within the Oxygen XML editor), and copy them in this folder before publishing the edition.
- `visCollImageList`. Indicate the path to the XML document holding the list of the manuscript images, default path is to `data/viscoll/[name of document]`. This document is not in TEI XML format, see VisColl documentation for more information.
- `visCollDataModel`. Indicate the path to the XML document holding the description of the manuscript quire structure, default path is to `data/viscoll/[name of document]`. This document is not in TEI XML format, see VisColl documentation for more information, the version required is the

data model 2.

View modes

- `defaultViewMode`. Select which view mode you want your edition to open on. Note that it must be an active mode!
- `availableViewModes`. Select which view modes you want to be available in your edition. You can deactivate a view mode either by deleting it or by setting the property `visible` to `false` (suggested method).

Edition levels

- `defaultEdition`. Select which edition level you want your edition to open on. Note that it must be an active edition level!
- `showEditionLevelSelector`. Decide if you want to display the edition level selector (`true`) or not `false`. This parameter is considered only if you select just one edition level: if there are two or more edition levels available, the edition selector will be always visible.
- `availableEditionLevel`. Select which edition levels you want to be available in your edition. You can deactivate a view mode both by deleting it and by setting to `false` the property `visible` (the latter being the suggested method). If you select just one edition level you can choose either to display the selector (with just one item) or not by setting `showEditionLevelSelector` respectively to `true` or `false`.

Edition navigation

- `showDocumentSelector`. Select if you want to activate (`true`) or not (`false`) the document selector, which allows the user to navigate an edition composed of different documents.
- `enableNavBar`. Select if you want to activate (`true`) or not (`false`) the navigation bar at the bottom of the screen.
- `initNavBarOpened`. If the navigation bar is active (see previous setting), sets its initial status: with `true` it will be shown when the edition is loaded, with `false` it will be hidden until the user decides to show it.
- `thumbnailsButton`. Select if you want to activate (`true`) or not (`false`) the thumbnail button in the navigation bar.
- `viscollButton`. Select if you want to activate (`true`) or not (`false`) the VisColl button in the navigation bar.

Generic tools

- `toolPinAppEntries`. Select if you want to activate (`true`) or not (`false`) the PIN tool, which allows the user to "save" apparatus entries or (named) entities in order to reach them more quickly when you need them.
- `toolImageTextLinking`. Select if you want to activate (`true`) or not (`false`) the Image-Text Linking tool, which allows the user to connect line by line the facsimile to the transcription. You will need to properly encode the `zone` and their coordinates and have Image-Text among the available view modes. Note that this is still a work-in-progress feature since we are still implementing the EVT 2 image viewer.

Named entities

- `namedEntitiesSelector`. Select if you want to activate (`true`) or not (`false`) the (named) entities selector, which allows the user to highlight on the text one or more specific (named) entitie(s).
- `namedEntitiesToHandle`. Customize the list of available named entities to be highlighted and to be shown among entities lists, by adding a new element in this list: for each element you should define a `tagName`, which is the XML tag that identifies the entity and a `label` that will be shown in the selector. If you don't need an entity that is already inserted in this list you can delete it or just use the property `available` set to `false` (suggested choice). Note that EVT can work properly only with `persName`, `placeName` and `orgName`; any other type of entity might cause problems (hopefully not!). If you need a new kind of named entity to be handled just notify the EVT Development Team.
- `otherEntitiesToHandle`. Customize the list of available entities to be highlighted by adding a new element in this list: for each element you should define a `tagName`, which is the XML tag that identify the entity, a `label` that will be shown in the selector and a `color` that will be used to highlight the entity within the text. If you don't need an entity that is already inserted in this list you can delete it or just use the property `available` set to `false` (suggested choice).

Critical edition

Witnesses

- `preferredWitness`. Indicate the sigla of your preferred witness; this will be used everywhere if you forgot to encode the lemma for a particular variation of the text.
- `skipWitnesses`. Indicate the siglas (divided by commas) of witnesses you want to exclude from visualization.

- `maxWitsLoadTogether`. Maximum number of witnesses which are going to be shown at the same time.

Witnesses Group(s)

- `witnessesGroups`. If you want, you can divide the readings of all critical apparatus entries into groups. Each group should have a `witnesses` property (mandatory) that indicates the siglas of witnesses within the group and a `groupName` (optional) that indicates the name of the group to be displayed (f.i. `{ "groupName": "Group 1", "witnesses": "A, B, B1" }`).

Apparatuses

EVT 2 is able to handle multiple levels of apparatuses: critical entries apparatus, sources apparatus and analogues apparatus. In "Reading view", all of them can be available both in inline mode (the apparatus will appear within the text, right after the portion of text to which it is connected) or in a separate box (there will be a container next to the main text where all the entries will be shown and aligned to the text, whenever the user clicks on an entry). By default, all the apparatuses will appear separately from the main text, but you can choose which mode you prefer by setting to `true` (inline) or `false` (separate box) the following parameters:

- `showInlineCriticalApparatus`, for critical apparatus entries;
- `showInlineSources`, for apparatus of sources;
- `showInlineAnalogues`, for apparatus of analogues.
- `showReadingExponent`, if you want to use an alphabetic exponent for critical entries (`true`) or not (`false`).

Tools

- `toolHeatMap`. Indicate if you want to include the Heat Map tool within the Critical Edition box (`true`) or not (`false`). This tool gives the user an overview about text variance.

Multiple recensions

- `versions`. Here you can specify the @xml:id values used to distinguish between different recensions (or versions), the first value entered corresponds to the version used as critical text for the edition.

Advanced Settings

Tell the system how to recognize the data: indicate which XML tag you used for the encoding of the different objects.

XML Tag usage configuration

- `listDef`. List of Witnesses: element(s) you used to encode the lists of all the witnesses or changes referred to by the critical apparatus (f.i. `<listWit>` or `<listChange>`). Please divide values using commas.
- `versionDef`. Single witness: element(s) you used to encode a single witness or change referred to within the critical apparatus (f.i. `<witness>` or `<change>`). Please divide values using commas.
- `fragmentMilestone`. Fragment milestones: element(s) you used to indicate the beginning (or resumption) and the end (or suspension) of the text of a fragmentary witness (f.i. `<witStart>` or `<witEnd>`). Please divide values using commas.
- `lacunaMilestone`. Lacuna milestones: element(s) you used to indicate the beginning and the end of a lacuna in the text of a mostly complete textual witness (f.i. `<lacunaStart>` or `<lacunaEnd>`). Please divide values using commas.
- `notSignificantVariant`. Not significant variants: element(s) of attribute(s) you used to encode variants that are not significant and you do not want to appear in the main critical apparatus (f.i. `<orig>`, `<sic>` or `@type=orthographic`). Please divide values using commas.
- `quoteDef`. Quotes: element(s) used within the XML file to encode quotes for the sources apparatus (f.i. `<quote>`). Please divide values using commas.
- `analogueDef`. Analogues: element(s) used within the XML file to encode passages for the analogues apparatus. (f.i. `<seg>` or `<ref[type=parallelPassage]>`). Please divide values using commas.

Filters

- `possibleLemmaFilters`. Possible lemma filters: attribute(s), divided by commas, you want to consider as possible filters for lemmas (f.i. `resp` or `cert`). If you want, you can customize the color of each filter value in the tab "Colors" (otherwise random colors will be used).
- `possibleVariantFilters`. Possible variant filters: attribute(s), divided by commas, you want to consider as possible filters for variants (f.i. `type`, `cause` or `hand`). If you want, you can customize the color of each filter value in the tab "Colors" (otherwise random colors will be used).

Colors

- `variantColorLight` and `variantColorDark`. Generic variant Colors: customize the highlight colors (both dark and light for selected and unselected entries) for generic variants that do not have any specific metadata (or have metadata that are not considered as filters). Default colors are `rgb(208, 220, 255)` (light) and `rgb(101, 138, 255)` (dark).

- `heatmapColor`. Heat Map Color: customize the highlight color for variants when the heat map tool is activated (this will be the darkest color possible, that means the color of entries with the highest variance level). The default color is `rgb(255, 108, 63)`.
- `variantColors`. Specific Variant Colors: customize the highlight color for each value of each lemma filter you defined in `possibleLemmaFilters` and each reading filter you defined in `possibleVariantFilters`. If you do not define a specific color, the system will use a random one.

Miscellaneous settings

Bibliography

- `defaultBibliographicStyle`. Select which bibliographic style you want your edition to open on. Note that it must be an active bibliographic style!
- `allowedBibliographicStyles`. Select which bibliographic style you want to enable. Bibliographic styles will work properly if the system will find all needed information encoded in you XML file. You can deactivate a bibliographic style either by deleting it or by setting the property `enabled` to `false` (suggested method).

Search engine

- `virtualKeyboardKeys`. Here you can specify special characters that will appear as buttons in a virtual keyboard tied to the search engine. To add a special character to the virtual keyboard write here the `@xml:id` value in the corresponding `<char>` or `<glyph>` element between quotation marks (e.g. `"amacr"` for an a with macron), separate values using commas.

Image viewer options

- `imageViewerOptions`. Miscellaneous options for the image viewer, modify only if you know what you're doing!
- `imageNormalizationCoefficient`. The value of this parameter must be that of the width of the manuscript/document images (in pixels), leaving the default value or setting a wrong value may lead to text-image links malfunctioning (specifically, drawing the wrong areas for linking on the images).

Languages

- `languages`. Customize the languages you want to set as available for the translation of the User Interface (just the UI!) by adding their codes in this list. At the moment we support just english ('en') and italian ('it'). If you want to add support for a new language, just add a new *new_language_code*.json inside the i18n directory and a *new_language_code*.png image inside the images folder.

2.3 CSS Customization

The customization of generic and linear TEI elements is very simple, even if EVT does not yet consider them in the default visualization: in fact, EVT transforms each XML element into an HTML <span> with a value for the class attribute corresponding to the TEI element name. TEI attributes remain as such, but are preceded by a data- prefix. In this way, if you want to visualize TEI elements which are not handled in any particular way by EVT, the customization is very easy: in the custom-style.css file that you will find in the config folder just add a rule that matches the tag name of the TEI element to style. To add a new style rule regarding a specific XML element you have to identify it first, for example if you want to add specific rules for <head> and <supplied>:

```
<head>
     <supplied resp="#silvia">Cap. I.5</supplied>
     ...
</head>
```

If we want to give a different style to the <head> and <supplied> elements, we will identify them as follows:

```
<span class="head">
  <span class="supplied" data-resp="#silvia">Cap. I.5</span>
      ...
</span>
```

After that we can create the style rules (CSS) for those elements: we need to open the config/custom-style.css file and declare the rules as follows:

```
.head {
   background-color: #CCCCCC;
 }

.supplied {
   color: #f6f6f6;
   background-color: #333;
 }

.supplied[data-resp="#silvia"] {
   background-color: #f2f2f2;
 }
```

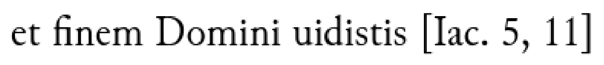
Note how we can select a specific rule for each `<supplied>` element whose encoder is Silvia. Once finished simply save the file and reload the EVT display page.

EVT has its own set of stylesheets, besides JavaScript-based processing, to handle the TEI elements needed to visualize the edition text. The custom CSS stylesheets can be particularly handy to handle elements which EVT knows nothing about:

```
.bibl::before {
     content: "[";
}

.bibl::after {
      content: "]";
}
```

These two simple rules will put the content of `<bibl>` elements between square brackets:


et finem Domini uidistis [Iac. 5, 11]

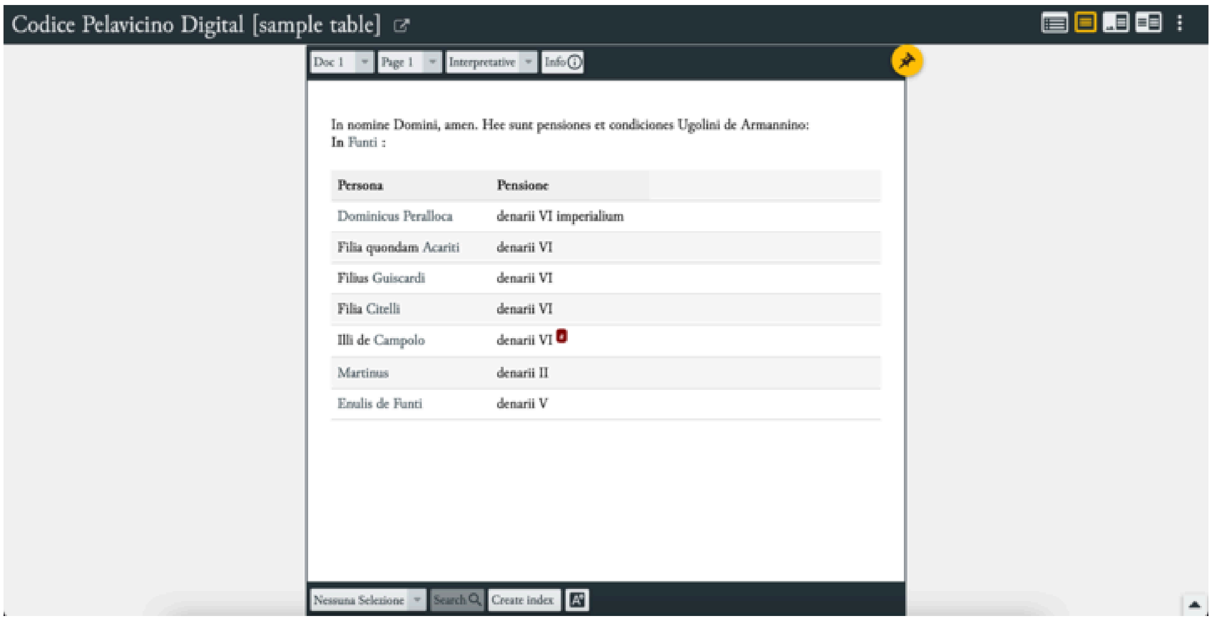
EVT doesn't handle `<table>` elements, but again here is CSS to the rescue:

```
.table {
     width: 100%;
     border-collapse: collapse;
}

.row {
     border-bottom: 1px solid #ccc;
     display: flex;
}

.cell {
     padding: 8px;
     text-align: left;
     width: 150px;
}
.cell[data-role="label"] {
     font-weight: bold;
     background-color: #f2f2f2;
}

.row:nth-child(even) {
     background-color: #f9f9f9;
}
```



In nomine Domini, amen. Hee sunt pensiones et condiciones Ugolini de Armannino:
In Funti :

Persona	Pensione
Dominicus Peralloca	denarii VI imperialium
Filia quondam Acariti	denarii VI
Filius Guiscardi	denarii VI
Filia Citelli	denarii VI
Illi de Campolo	denarii VI
Martinus	denarii II
Emulis de Funti	denarii V

Please note that CSS customization works better for elements not handled by EVT, and for general text display: if an element is already processed by EVT and/or is shown in a dedicated container (f.i. a critical apparatus entry) it is likely that this customization won't work. Trying doesn't cause problems, anyway, you may also add **`!important`** to the rule, e.g.:

```
.head[data-type="main"] {
 font-style: italic !important;
 text-align: center !important;
 font-size: 200% !important;
}
```

3. TEI encoding of textual data

EVT 2 supports both diplomatic transcriptions with the corresponding manuscript images, in a way similar to the previous version (EVT 1), and critical editions encoded according to the parallel segmentation method (experimental support for the double-end-point attached method is under way and will be added to a future version).

3.1 TEI Header

The `<teiHeader>` element at the top of the XML file must be compiled according to the standard TEI *Guidelines*, providing all the necessary information about the project and its curators. Note that if you decide to include one or more `<msDesc>` elements these will be used to show information pertaining to the corresponding manuscripts in the image frame, where the digitized manuscript images are shown.

Please also note that if you wish to take advantage of the support for named entities, this is where you should insert the lists (e.g. `<listPerson>`), this is explained in more detail in the *2.4 Support for named entities* section.

3.2 Text structure

Documents parsed by EVT must conform to the TEI *Guidelines* with regard to the general text structure of a work:

- **`<text>`** follows the **`<facsimile>`** element (if present) and holds the following items:
  - `<front>` front matter (information about a text, regesto (diplomatic text summary), etc.)
  - `<body>` actual text of the work encoded
  - `<back>` back matter (notes, comments, etc.)

- EVT also support the **`<group>`** element, so that you can have more than one <text> in a single TEI document, which is perfect for miscellaneous manuscripts:

```
<text>
  <front> [ front matter for the whole document ] </front>
  <group>
     <text>
        <front> [ front matter for the first text ] </front>
        <body> [ body of the first text ]           </body>
        <back> [ back matter for the first text ]   </back>
     </text>
     <text>
        <front> [ front matter for the second text] </front>
        <body> [ body of the second text ]          </body>
        <back> [ back matter for the second text ]  </back>
     </text>
        ... [ more texts ] ...
  </group>
  <back> [ back matter for the whole document ] </back>
</text>
```

  Note that you can also **XInclude** to keep the texts in separate TEI documents:

```
<text>
  <group>
```

```
    <xi:include href="vb/VB-21-SoulI.xml"
        xmlns:xi="http://www.w3.org/2001/XInclude"
        xpointer="VB_text_SoulI"/>

    <xi:include href="vb/VB-23-DOTR.xml"
        xmlns:xi="http://www.w3.org/2001/XInclude"
        xpointer="VB_text_DOTR"/>

  </group>
</text>
```

where the `xpointer` attribute holds the values of the `xml:id` for the `<text>` element of the file specified in the `href` attribute.

- At the present moment, `<front>` content is limited to new, editor-curated introductory material. `<titlePage>` and original text are not supported because all the content of the `<front>` element will be displayed in the "Info" frame related to a single text. This will be fixed in the following version.
- Within `<body>`, text can be distributed into `<div>` elements which are then used to navigate it, a navigation selector widget will be created automatically to show the different parts of a work. The visible navigation labels are created according to this scheme:
    - if the `<div>` element has a `@type` attribute, its value is saved and will be shown with an uppercase first letter; otherwise a default `Div.` text will be shown;
    - if the `<div>` element has a `@subtype` attribute as well, its value is going to be added to the preceding separated by a `"-"`, again with an uppercase first letter;
    - a blank space will be added after the preceding;
    - after which the content of `@n` will be added, if this attribute has been used, otherwise the number of the `<div>` parsed up until the current one plus 1.

So if only `@n` is used the resulting text division name in the navigation UI will be `Div. 1`.

3.3 Diplomatic edition

3.3.1 Facsimile

The `<facsimile>` element is where all the information necessary for image-text linking is saved, both for hot-spot and for continuous linking (e.g. page or line level) purposes. Note that you can do without this element in the transcription document, EVT will work and show your texts at page level linking without the need of having a `<facsimile>` (see below instructions about `<pb/>`) or even if there are no images at all: in the latter case, to avoid having an empty image frame just set the corresponding view mode (`"label": "IMAGE_TEXT"`) to `false`.

A `<facsimile>` holds as many `<surface>`s as there are single side images. Each `<surface>` may include:

- a `corresp` attribute pointing to the `xml:id` of the corresponding `<pb/>` element

  ```
  <surface xml:id="VB_surf_104v" corresp="#VB_fol_104v">
  ```

- a `<graphic>` element: this can include an `url` attribute pointing to a specific file location, and other attributes, but at the present moment this is not used by EVT;

  ```
  <graphic width="1200px" height="1800px"
           url="../images/Vercelli-Book_104V_S_300dpi.jpg"/>
  ```

- a number of `<zone>` elements: these are optionally used to record the coordinates of image areas corresponding to text elements and other metadata information, to do so they require use of several attributes:

◦ **ulx**, **uly**, **lrx**, **lry**	cartesian coordinates of the image area
◦ **rendition**	two possible values: **Line** (for line-by-line text-image linking) and **HotSpot** (for image areas to be used in hotspots)
◦ **xml:id**	the unique ID for the current `<zone>`
◦ **corresp**	points to the corresponding `<lb/>` ID (i.e. line) in the text transcription (OPTIONAL)

  ```
  <zone corresp="#VB_lb_104v_01" lrx="1052" lry="211"
        rend="visible" rendition="Line" ulx="261" uly="156"
        xml:id="VB_line_104v_01"/>
  ```

  Complete example:

```
<facsimile xml:id="VB_fac_dotr">
  <surface xml:id="VB_surf_104v" corresp="#VB_fol_104v">
     <graphic height="1800px" width="1200px"
              url="immagini\Vercelli-Book_104V_S_300dpi.jpg"/>
     <zone corresp="#VB_lb_104v_01" lrx="1052" lry="211"
```

```
          rend="visible" rendition="Line" ulx="261" uly="156"
          xml:id="VB_line_104v_01"/>
     <zone corresp="#VB_lb_104v_02" lrx="1072" lry="263"
          rend="visible" rendition="Line" ulx="257" uly="209"
          xml:id="VB_line_104v_02"/>
      [...]
     <zone corresp="#VB_lb_104v_23" lrx="1084" lry="1318"
          rend="visible" rendition="Line" ulx="262" uly="1269"
          xml:id="VB_line_104v_23"/>
     <zone corresp="#VB_lb_104v_24" lrx="1104" lry="1372"
          end="visible" rendition="Line" ulx="260" uly="1316"
          xml:id="VB_line_104v_24"/>
  </surface>
```

For more information about digital facsimiles in TEI XML see chapter 11 Representation of Primary Sources of the TEI *Guidelines*.

3.2.2 Parallel Transcription

The Parallel Transcription method is the most popular and recommended way to couple a (semi-)diplomatic transcription with digitized manuscript images. This is what EVT expects to find in a TEI XML document created according to this method:

- To provide a safe starting point for the parsing of the TEI document it is essential that each `<text>` includes an **`xml:id`** attribute holding an unique ID for the text: if you forget to include this attribute EVT will create one automatically, but it is surely preferable to have it, possibly within a project-wide naming schema, from the start for all your `<text>` elements. Also, while not mandatory, it is highly recommended to add an **`n`** attribute holding the text title, so that it can be showed in the appropriate selector in the web edition; if `n` is not available, `xml:id` will be used removing the underscore characters (in other words, results may be acceptable but not always pretty …). Other elements may be used as necessary per specific needs, f.i. `type` to record if the text is in prose or verse, but they aren't relevant for EVT. Example:

  ```
  <text n="The Dream of the Rood" type="verse"
        xml:id="DOTR">
  ```

- **`<pb/>`** elements are used to mark up folio sides: they must include the `n` and `xml:id` attributes, respectively to show the correct folio number and to enable text-image linking at the page level; more in detail, the **`n`** is the label you will see in the page selector in the web interface, so you are relatively free to choose the characters that you put in it, while the **`xml:id`** is a unique identifier used to recover the text and image of each page: you can't have two identical values for `xml:id` in your file, while you can have two or more identical `n` attributes. For the image-text linking at page level to work properly, it is necessary to add a **`facs`** attribute pointing to the corresponding image file:

```
<pb n="104v" xml:id="VB_fol_104v"
    facs="data/images/single/VB_fol_104v.jpg"/>
```

Important: at the present moment, when creating a diplomatic edition and selecting the diplomatic and/or interpretative edition levels as available (see above the configuration section) it is mandatory to have at least a `<pb/>` at the beginning of the transcription.

- **`<lb/>`** elements are used to mark up manuscript lines: they must include the `n` and `xml:id` attributes, respectively for line numbering and text-image linking at the line level; `xml:id` only serves the purpose of text-image linking so you can omit it if you're not going to take advantage of this feature.

  ```
  <lb facs="#VB_line_104v_07" n="7"
      xml:id="VB_lb_104v_07"/>
  ```

  `<lb/>`s with no attributes (or an `<lb rend="empty"/>` element) will be rendered as empty lines. Note that you can have numbered empty lines, but again this works if you add the `n` attribute:

  ```
  <lb n="1"/>
  <lb n="2"/>
  <lb n="3"/>
  ```

3.3.3 Support for named entities

Support for named entities is a feature originally introduced in EVT 1 and now available in EVT 2 as well. To best take advantage of this feature, it is necessary to prepare lists that will be included in an appropriate location, at the present moment EVT supports the `<sourceDesc>` element within the `<teiHeader>`.

One likely candidate is a list of persons, as in this example (selected from the Codice Pelavicino digital edition):

```
<listPerson>
  <person xml:id="AccursetusVitalis">
```

```
      <persName>
        <forename>Accursetus</forename>
        <surname>quondam Vitalis de castro Sarzane</surname>
      </persName>
      <sex>M</sex>
    </person>
    <person xml:id="AcoltusBonavite">
      <persName>
        <forename>Acoltus</forename>
        <surname>quondam Bonavite de Trebiano</surname>
      </persName>
      <sex>M</sex>
    </person>
    <person xml:id="Adiutus">
      <persName>
        <forename>Adiutus/Aiutus</forename>
      </persName>
      <sex>M</sex>
    </person>
    <person xml:id="Adiutuscap">
      <persName>
        <forename>Adiutus / Aiutus</forename>
      </persName>
      <sex>M</sex>
      <occupation n="2">presbiter, capellanus domini Guilielmi</occupation>
    </person>
    <person xml:id="Adiutusnot">
      <persName>
        <forename>Adiutus / Aiutus</forename>
      </persName>
      <sex>M</sex>
      <occupation n="1">sacri palatii notarius</occupation>
    </person>
    <person xml:id="Adornellusnot">
      <persName>
        <forename>Adornellus</forename>
        <surname>domini Tranchedi comitis de Advocatis de Luca</surname>
      </persName>
      <sex>M</sex>
      <occupation n="2">iudex ordinarius, notarius</occupation>
    </person>
  [...]
  </listPerson>
```

Note that every `<person>` item relates to an individual, in this case historical figures, but they might as well be fictional characters. Another useful list may one of places (again from the CPD edition):

```
<listPlace>
  <place xml:id="Aciliano">
    <settlement type="località">Aciliano</settlement>
  </place>
  <place xml:id="Acola">
    <settlement type="località">Acola</settlement>
  </place>
  <place xml:id="Agina">
    <settlement type="località">Agina</settlement>
  </place>
  <place xml:id="Alione">
    <settlement type="località">Alione</settlement>
  </place>
  <place xml:id="Alpes">
    <settlement type="monte">Alpes</settlement>
    <placeName type="new">Alpi Apuane</placeName>
  </place>
[...]
<listPlace>
```

At the present moment, EVT supports the following list elements for the purpose of named entity annotation and linking:

- `<listPerson>`
- `<listPlace>`
- `<listOrg>`

Note that the TEI schemas offer other specific list elements, such as `<listNym>` for the canonical form of names, or `<listEvent>` for historical events, but it is also possible to use the generic `<list>` element to provide for other needs (you only have to specify a value for `type` to make explicit what kind of items are gathered in a list), provided that all the lists are related to individual "objects". Future versions of EVT may support other types of list, including the generic one, but at the moment only those specified above are supported.

Once the lists are ready, you can link each occurrence of an item to the corresponding list entry using the `ref` attribute for elements such as `<persName>` (→ `<listPerson>`) and `<placeName>` (→ `<listPlace>`). As a value for each `ref` you will have to use the `xml:id` of the corresponding item in a list:

```
<p xml:id="CCLXXXXII_p_003" n="3"><ptr target="CCLXXXXII_st_001"
facs="#st_279r_001"/> Ego <persName ref="#Adiutusnot">Adiutus,
<roleName>sacri palacii notarius</roleName></persName>, hiis interfui et
de mandato suprascripti domini episcopi hanc cartam abreviavi et in
publicam
formam redegi, signum et nomen proprium apponendo.</p>
```

Edition levels

Different **edition levels** in the same TEI document are managed through a combination of transcriptional and editorial elements:

- the **diplomatic** level is encoded using the `<damage>`, `<hi>`, `<abbr>`+`<am>` and `<orig>` elements; at the character level, if a `<charDecl>` is present, using the `<mapping type="diplomatic">` character values inside each <char> (or <glyph>) element;
- the **interpretative** level is encoded using the `<supplied>`, `<expan>`+`<ex>` and `<reg>` elements; at the character level, if a `<charDecl>` is present, using the `<mapping type="normalized">` character values inside each `<char>` (or `<glyph>`) element; `<sic>`/`<corr>` pairs are possible, but not necessary if a full critical edition is envisaged.

Note that all the editorial elements are usually inserted in <choice> elements: this is mandatory for word-level pairs.

Full example:

```
<text>
 <body>
   <div n="DOTR" subtype="edition_text" type="verse" xml:id="DOTR">
    <pb n="104v" xml:id="VB_fol_104v"/>
     <l n="1"><lb facs="#VB_line_104v_07" n="7" xml:id="VB_lb_104v_07"/>
           <hi rend="init3.1"><g ref="#Hunc"/></hi>
           <hi rend="cap">W</hi>æt ic
           <g ref="#slong"/>wefna c<g ref="#ydot"/>
           <g ref="#sins"/>t secgan wylle</l>
     <l n="2"><choice>
           <sic>hæt</sic>
                 <corr resp="Grein">hwæt</corr>
              </choice>
               <choice>
                 <orig>mege mætte</orig>
                 <reg>me gemætte</reg>
               </choice>
           <lb facs="#VB_line_104v_08" n="8" xml:id="VB_lb_104v_08"/>
           to midre nihte</l>
            <l n="3">syðþan <choice>
                 <orig>reord b<g ref="#eenl"/>r<g ref="#eenl"/>nd</orig>
                 <reg>reordberend</reg>
               </choice> reste wunedon<orig><pc
```

```
        type="metrical">.</pc></orig></l>
                <l n="4"><lb facs="#VB_line_104v_09" n="9"
        xml:id="VB_lb_104v_09"/>þuhte me þæt ic <choice>
                    <orig>ge <g ref="#sins"/>awe</orig>
                    <reg>gesawe</reg>
                  </choice>
                  <g ref="#sins"/>yllicre treow</l>

        [...]

            <l n="156">ælmihtig <name type="religion"><choice>
                <orig>god</orig>
                <reg>God</reg>
              </choice></name> þær hi<g ref="#sins"/> eðel wæ<g ref="#sins"/>
            <g ref="#colmidcomposit"/></l>
        </div>
      </body>
    </text>
```

3.4 Critical editions

EVT accepts critical editions encoded according to the parallel segmentation method. Initial support for the double-end-point attached method has been recently added, but it is very much experimental at this moment.

See section 2.2 for a description of the configuration options related to critical editions.

4. Other features

4.1 VisColl: describing and visualizing the manuscript quire structure

EVT 2 beta2 introduces a new and very interesting feature: support for VisColl, a software tool aiming at building models of the physical collation of manuscripts, and then visualizing them in various ways. While not based on the TEI schemas and guidelines, VisColl is based on XML documents to describe the manuscript quire structure and on XSLT style sheets to create dynamic representations of such structure using SVG pictures combined with manuscript folios miniatures. Since VisColl is open source software, the EVT development team managed to adapt the XSLT style sheets so that they could be integrated in EVT build chain.

To show the quire structure of a manuscript using VisColl and EVT you have to prepare two different files:

- an XML file describing such structure according to the VisColl data model 1 (Collation Model);
- an XML file holding a list of the manuscript images (Image List).

Both quire description and image list can be prepared by hand, VisColl makes use of a custom XML schema which is quite simple, see the examples below. As an alternative, you can follow the instructions available on its home page, see the How To page.

This is an example of a manuscript quire description (Collation Model) based on the Vercelli Book codex:

```
<quires>
	<quire n="1" xml:id="VB-q-1">1</quire>
	<quire n="2" xml:id="VB-q-2">2</quire>
	<quire n="3" xml:id="VB-q-3">3</quire>
	<quire n="4" xml:id="VB-q-4">4</quire>
	<quire n="5" xml:id="VB-q-5">5</quire>
	<quire n="6" xml:id="VB-q-6">6</quire>
	<quire n="7" xml:id="VB-q-7">7</quire>
	<quire n="8" xml:id="VB-q-8">8</quire>
	<quire n="9" xml:id="VB-q-9">9</quire>
	<quire n="10" xml:id="VB-q-10">10</quire>
	<quire n="11" xml:id="VB-q-11">11</quire>
	<quire n="12" xml:id="VB-q-12">12</quire>
	<quire n="13" xml:id="VB-q-13">13</quire>
	<quire n="14" xml:id="VB-q-14">14</quire>
	<quire n="15" xml:id="VB-q-15">15</quire>
	<quire n="16" xml:id="VB-q-16">16</quire>
	<quire n="17" xml:id="VB-q-17">17</quire>
	<quire n="18" xml:id="VB-q-18">18</quire>
	<quire n="19" xml:id="VB-q-19">19</quire>
</quires>

[...]

<leaf xml:id="VB-14-1">
	<folioNumber certainty="1" val="99">99</folioNumber>
	<mode certainty="1" val="original"></mode>
	<q leafno="1" n="14" position="1" target="#VB-q-14">
	<conjoin certainty="1" target="#VB-14-8"></conjoin>
	</q>
</leaf>
<leaf xml:id="VB-14-2">
	<folioNumber certainty="1" val="100">100</folioNumber>
	<mode certainty="1" val="original"></mode>
	<q leafno="2" n="14" position="2" target="#VB-q-14">
	<conjoin certainty="1" target="#VB-14-7"></conjoin>
	</q>
</leaf>
```

```
<leaf xml:id="VB-14-3">
     <mode certainty="1" val="missing"></mode>
     <q leafno="3" n="14" position="3" target="#VB-q-14">
     <conjoin certainty="1" target="#VB-14-6"></conjoin>
     </q>
</leaf>
<leaf xml:id="VB-14-4">
     <folioNumber certainty="1" val="101">101</folioNumber>
     <mode certainty="1" val="original"></mode>
     <q leafno="4" n="14" position="4" target="#VB-q-14">
     <conjoin certainty="1" target="#VB-14-5"></conjoin>
     </q>
</leaf>
<leaf xml:id="VB-14-5">
     <folioNumber certainty="1" val="102">102</folioNumber>
     <mode certainty="1" val="original"></mode>
     <q leafno="5" n="14" position="5" target="#VB-q-14">
     <conjoin certainty="1" target="#VB-14-4"></conjoin>
     </q>
</leaf>
<leaf xml:id="VB-14-6">
     <folioNumber certainty="1" val="103">103</folioNumber>
     <mode certainty="1" val="original"></mode>
     <q leafno="6" n="14" position="6" target="#VB-q-14">
     <conjoin certainty="1" target="#VB-14-3"></conjoin>
     </q>
</leaf>
<leaf xml:id="VB-14-7">
     <mode certainty="1" val="missing"></mode>
     <q leafno="7" n="14" position="7" target="#VB-q-14">
     <conjoin certainty="1" target="#VB-14-2"></conjoin>
     </q>
</leaf>
<leaf xml:id="VB-14-8">
     <folioNumber certainty="1" val="104">104</folioNumber>
     <mode certainty="1" val="original"></mode>
     <q leafno="8" n="14" position="8" target="#VB-q-14">
     <conjoin certainty="1" target="#VB-14-1"></conjoin>
     </q>
</leaf>

[...]
```

While this is the corresponding image list:

```
<imageList>
[...]
     <imagerv>
     <image val="104v" url="data/images/single/VB_fol_104v.jpg"
     id="VB-14-8-v">104v</image>
```

```
        </imagerv>
        <imagerv>
        <image val="105r" url="data/images/single/VB_fol_105r.jpg"
        id="VB-15-1-r">105r</image>
        <image val="105v" url="data/images/single/VB_fol_105v.jpg"
        id="VB-15-1-v">105v</image>
        </imagerv>
        <imagerv>
        <image val="106r" url="data/images/single/VB_fol_106r.jpg"
        id="VB-15-2-r">106r</image>
        </imagerv>
[...]
</imageList>
```

The quire structure description (e.g. `VB-dataModel.xml`) and the image list (e.g. `VB-imageList.xml`) should be copied into the `data/viscoll` directory and the complete path has to be specified in the configuration file setting the corresponding parameters, `visCollDataModel` and `visCollImageList` (see above).

When the relevant XML files are in the appropriate place, it is necessary to execute the VisColl stylesheet by means of an XSLT 2 processor, so that you will get the SVG diagrams automatically and the VisColl button in the navigation bar when the Web edition is generated. If using the Oxygen XML editor, to start the transformation it is necessary to configure a transformation scenario providing in input the XML file (e.g. `VB-dataModel.xml`) and choosing the XSLT 2 stylesheet (`collationXMLtoSVGmod2.xsl`), and setting the output folder (choose `data/viscoll/SVG`). Then using the "Apply selected scenarios" to execute the transformation and generate all the SVG images in the specified folder.

Besides showing the physical structure of a manuscript binding, VisColl images can be used as an alternative navigation tool: if you click on a thumbnail, you will be carried to that manuscript folio image.

4.2 3DHOP

In EVT 2 beta 2 the 3DHOP (3D Heritage Online Presenter) software has been integrated so that it is now possible to load a 3D model into a dedicated frame and confront it with a description and/or other related text. This is an experimental feature which will be refined adding 3D model - edition text linking in the future.

To take advantage of this feature you will have to copy the 3D model files in the appropriate folders (`data/models/multires` and `data/models/singleres`, visit the 3DHOP home page to see which formats are supported) and configure EVT accordingly (see the `RC.xml` document and its `config_visionaryCross.json` settings file for a sample edition).

When testing a local instance, we recommend to use Chrome with the local file access parameter to use the 3DHOP view.

5. Sample editions

There are several ready-to-use examples. The one used by default is n. 1.

If you want to explore the other two you will just have to open the corresponding settings file (f.i. `config_marlowe.json`) and save it as the main `config.json` file, overwriting the existing configuration.[2] Then go to the `index.html` opened in your browser and reload the page!

- EXAMPLE 1: `avicenna.xml` - Short extract of *Edizione Logica Avicennae*, changed by CM for EVT testing purposes. It presents multiple levels of apparatuses (critical entries, sources and analogues), displayed in a separate dedicated frame. Configuration file for this edition: `config_avicenna.json`.
- EXAMPLE 2: `pseudoEdition.xml` - Pseudo edition for demonstration and testing purposes only, originally encoded by Marjorie Burghart for her TEI Critical Toolbox software, and modified in order to cover the highest number of possible use cases. It presents just the critical apparatus entry, displayed inline, within the main text. Configuration file for this edition: `config_pseudoEdition.json`.
- EXAMPLE 3: `pelavicino.xml` - Short extract of the Codice Pelavicino edition, which presents the encoding of named entities, in particular person, place and organization names. Configuration file for this edition: `config_pelavicino.json`.
- EXAMPLE 4: `marlowe.xml` - Short extract of *The Tragedie of Doctor Faustus* (B text) by Christopher Marlowe. Text provided by Perseus Digital Library, with funding from Tufts University. Original version available for viewing and download at http://www.perseus.tufts.edu/hopper/. Configuration file for this edition: `config_marlowe.json`.
- EXAMPLE 5: `RC.xml` - An experimental edition of the Ruthwell Cross to test the 3DHOP functionality recently added to EVT. Configuration file for this edition: `config_visionaryCross.json`.

To try the different sample editions you have to load the correspondent config file (e.g. `config_pelavicino.json`) in your editor, then save it (generally there is a `Save as…` item in the `File` menu of any text editor) as `config.json`. Don't worry if this means that

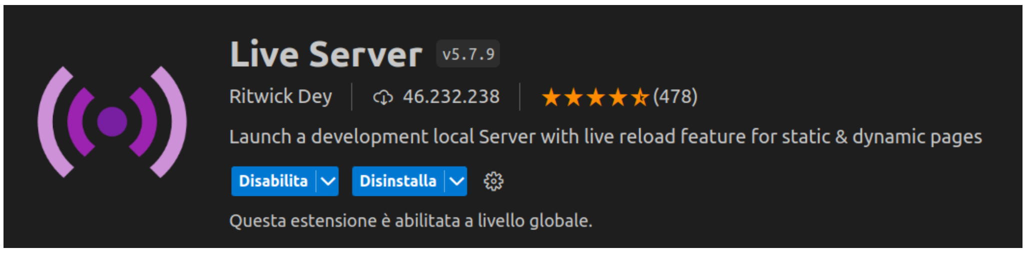
[2] Note that EVT only looks at `config.json` to load settings, all other files in the `config` folder are ignored and are used to make available ready-to-use configuration for the sample edition files.

you are going to overwrite the existing `config.json` file, as long as you don't delete the sample edition-related files you will always have a way to go back to the desired edition.

5.1 A note about browsers for local testing

Starting with version 67, Firefox developers decided to follow the same cautious policy already adopted by Chrome (and other browsers), i.e. to prevent the loading of local resources by JavaScript programs running in the browser. While this policy guarantees more security for the user, unfortunately it has the unpleasant side effect of preventing local loading of editions based on EVT (which is implemented in JavaScript) or other similar programs. In other words, while before v. 67 you could just load your local `index.html` file in Firefox and browse an EVT edition on your computer, which is very convenient to check for progress and fixes, now that is not possible anymore (when using Chrome, Opera, Edge etc. as well). Fortunately there are some solutions that can be adopted to do the local testing of an EVT-based edition:

- if you use **Visual Studio Code** as an XML editor, but also if you use Oxygen or something else, a very effective method is based on the **Live server extension**:

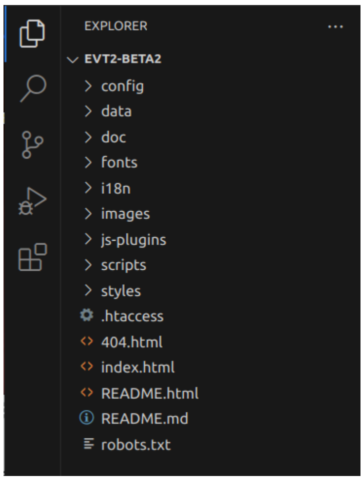


Live Server v5.7.9
Ritwick Dey | 46.232.238 | ★★★★★ (478)
Launch a development local Server with live reload feature for static & dynamic pages

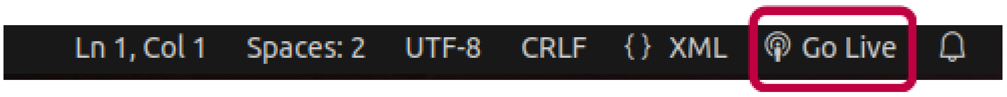
Questa estensione è abilitata a livello globale.

  after installing it, to load the edition into the browser you have to

  - select the EVT folder: menu `File → Open folder` open the EVT folder after having unpacked the `materiali.zip` archive, you should be able to see the `index.html` file in the panel on the left:



- click on the `Go Live` button in the status bar, bottom right corner:



- launch Chrome from the **command line** with the parameter `--allow-file-access-from-files`:

```
chrome.exe --allow-file-access-from-files
```

or (two separate lines)

```
cd "C:\Program Files (x86)\Google\Chrome\Application\"
  .\chrome.exe --allow-file-access-from-files
```

then press CTRL+O to open the index.html file in the EVT folder, or directly drag&drop it on the Chrome window. On a Mac the corresponding instruction is

```
open /Applications/Google\ Chrome.app --args
--allow-file-access-from-files
```

or

```
open -a "Google Chrome" --args --allow-file-access-from-files
```

- install an extension that acts as a local web server on Firefox or Chrome, e.g. this one for Chrome:

  https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb

- download and install **Firefox ESR** version **60**: this version, now very much outdated, predates the introduction of this new security policy and you can install it in parallel with other versions of Firefox (NB: it is **not** recommended to us this version for your usual browsing needs);
- install the development environment following the instructions on the GitHub repository (use the develop branch), an option recommended only for those familiar with git and GitHub-based development: https://github.com/evt-project/evt-viewer-angular

In any case, the problem only concerns local testing of your TEI documents in EVT, once the final edition is loaded on a web server there are no problems in accessing it with any of the most popular browsers.

Feedback

User feedback is very much appreciated: please send all comments, suggestions, bug reports, etc. to evt.developers@gmail.com.

References

Di Pietro, Chiara, and Roberto Rosselli Del Turco. “Between Innovation and Conservation: The Narrow Path of User Interface Design for Digital Scholarly Editions.” *Digital Scholarly Editions as Interfaces*, vol. 12, BoD, 2018, pp. 133–63. *kups.ub.uni-koeln.de*, https://kups.ub.uni-koeln.de/9085/.

Monella, Paolo, and Roberto Rosselli DelTurco. “Extending the DSE: LOD Support and TEI/IIIF Integration in EVT.” *Umanistica Digitale*, 2020, p. 10. http://amsacta.unibo.it/6316/.

Rosselli Del Turco Roberto. *EVT Development: An Update (and Quite a Bit of History)*. 2014, https://visualizationtechnology.wordpress.com/2014/01/26/evt-development-an-update-and-quite-a-bit-of-history/.

Roberto Rosselli Del Turco, et al. “Edition Visualization Technology: A Simple Tool to Visualize TEI-Based Digital Editions.” *JOURNAL OF THE TEXT ENCODING INITIATIVE*, no. Issue 8, 2015, pp. 1–21, doi:10.4000/jtei.1077.

Rosselli Del Turco, Roberto, et al. “Progettazione e implementazione di nuove funzionalità per EVT 2: lo stato attuale dello sviluppo.” *Umanistica Digitale*, no. 7, 2019, pp. 5–21. *umanisticadigitale.unibo.it*, doi:10.6092/issn.2532-8816/9322.

Rosselli Del Turco, Roberto. “Designing an Advanced Software Tool for Digital Scholarly Editions:” *Textual Cultures*, vol. 12, no. 2, Aug. 2019, pp. 91–111. *DOI.org (Crossref)*, doi:10.14434/textual.v12i2.27690.