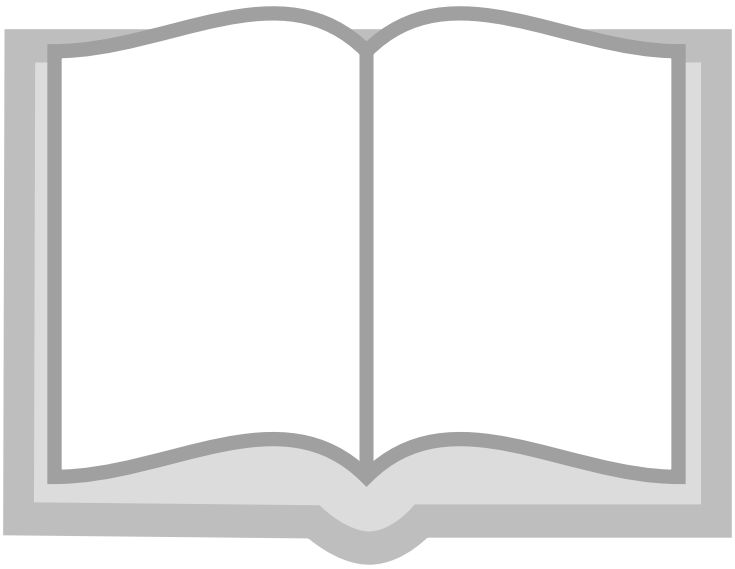
Publication Chain 1.0

Admin and User Guide

How to setup and use a publication chain



Xavier Berger

Abstract

This article is describing how to install and use a publication chain based on publican and serna-free software. This article has been written using this publication chain and is used as an example of document redaction and publication.

About the author

[1] http://creativecommons.org/licenses/by-sa/3.0/

Xavier Berger is working as Solution Architect in a telecom company. He is a specialist in Linux and network deployment. Xavier enjoys hiking, geocaching, skiing and spending time with his family. His web site is: *http://xberger.free.fr*

1. Introduction 2
2. Installation 5
    2.1. Install publican and poedit 5
    2.2. Install serna-free 5
3. Redaction 6
    3.1. Creating the article's skeleton 6
    3.2. Preparing the document structure 9
    3.3. Writing the article 10
4. Publication 14
    4.1. Rendering the document 14
    4.2. Managing a web site 15
5. Translation 17

A. Revision History **19**

1. Introduction

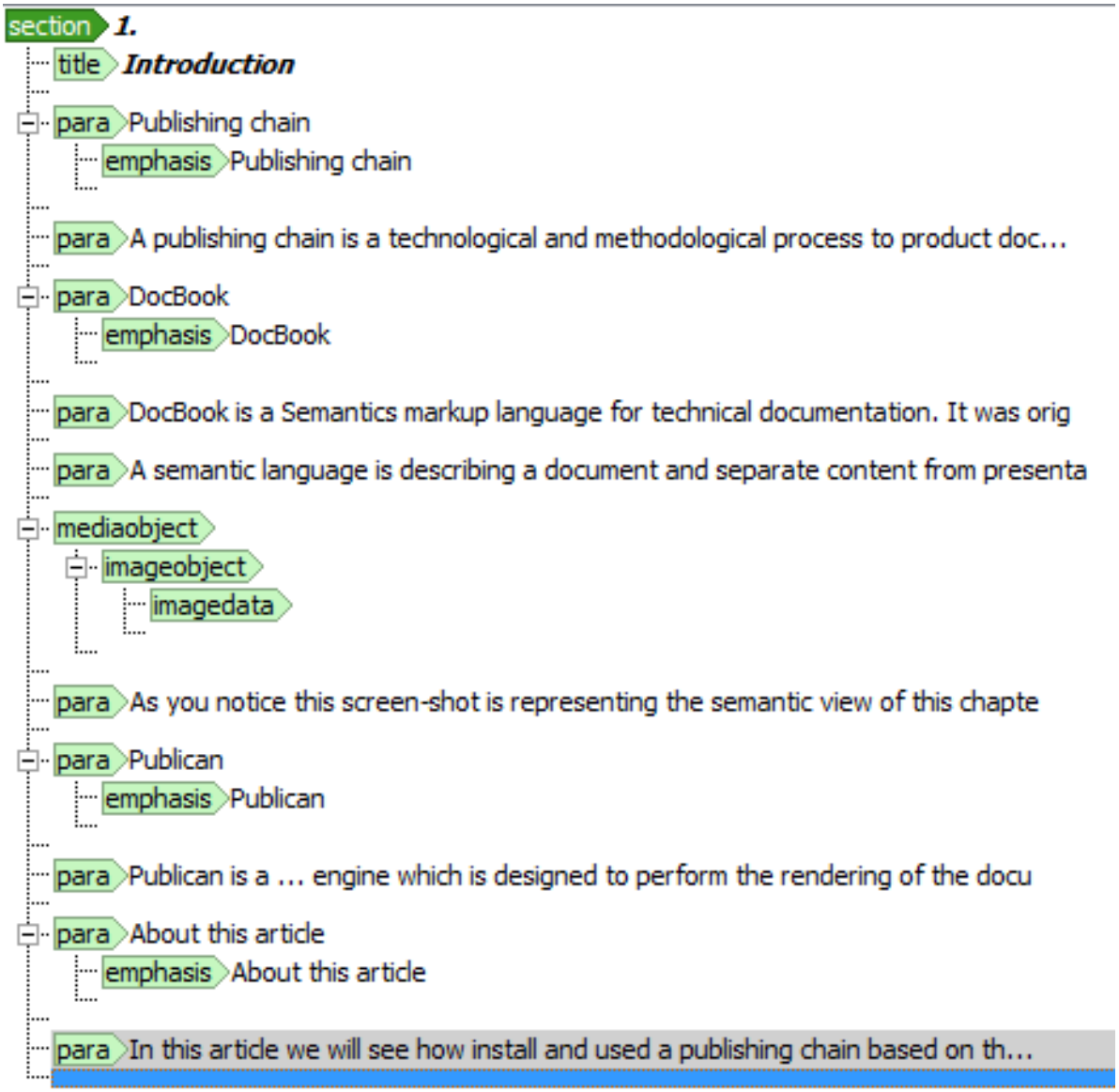
Publication chain

A publication chain is a technological and methodological process to product documents. The publication chain approach is to create a document model containing the data and the description of the formatting. The final document is generated by rendering in the desired format (pdf, html, epub...)

DocBook

DocBook is a semantics markup language for technical documentation. It was originally intended for writing technical documents related to computer hardware and software but it can be used for any other sort of documentation.

A semantic language is describing a document and separate content from presentation. **DocBook** file is storing the data XML format. The following screenshot is showing how a document described with a semantic markup language could be represented.

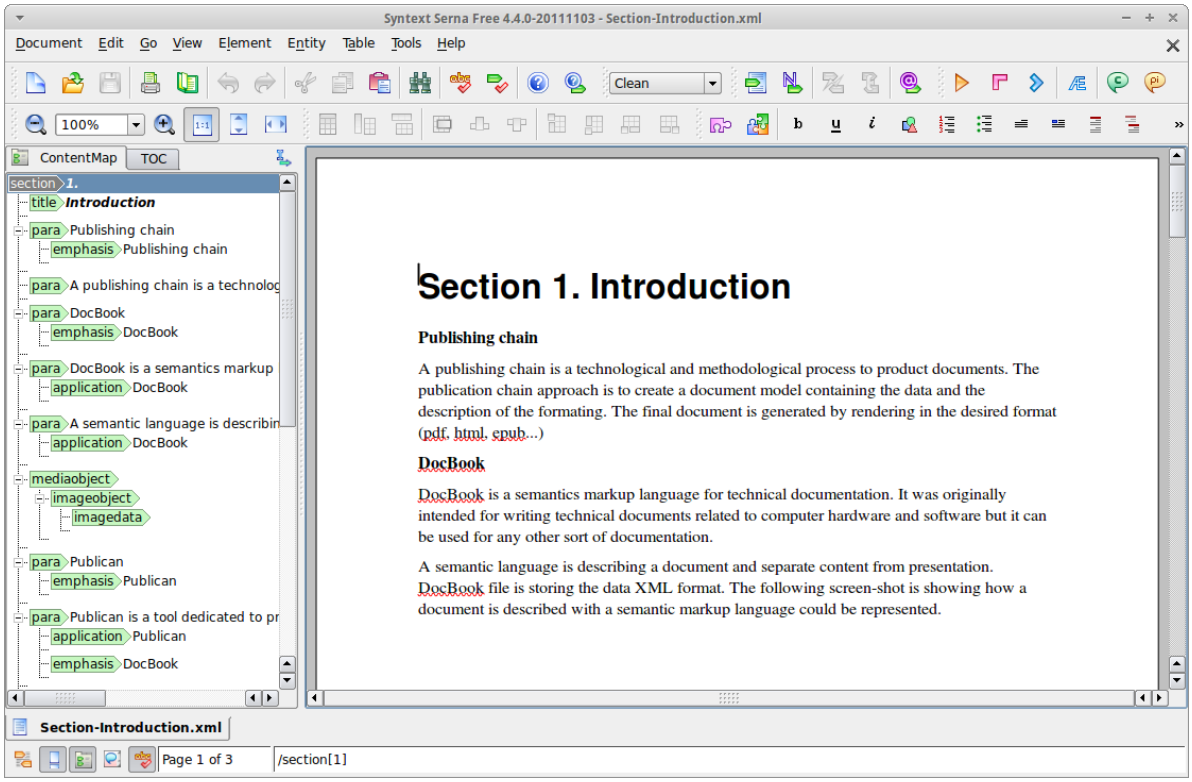


Publican

Publican is a tool dedicated to process **DocBook** XML. It is a rendering engine which can generate document in different kind of format by applying a transformation to the XML data. It will result reader friendly documents in various format like pdf, html or ePub.

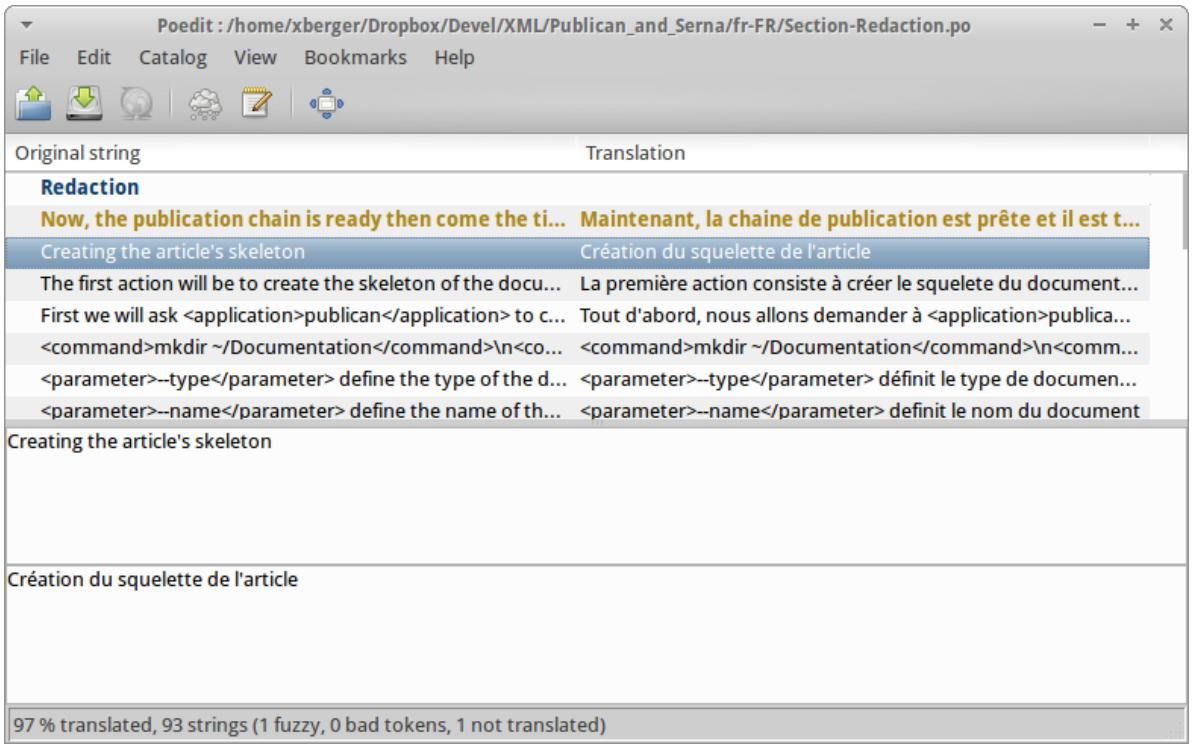
Serna

serna-free is an open source software designed to manipulate XML data in various format. **DocBook** is one of these formats. **serna-free** provides a WYSIWYM GUI: What You See Is What You Mean Graphical User Interface. It allows to easily edit XML documents.



poedit

poedit is a software dedication to the translation. It give to the translator a graphical user interface and shows the status of the translation of the document. We will see in the last part of this document how to use it and provide a multi-language documentation.



About this article

This article has been written using **`serna-free`**, **`publican`** and **`poedit`**. We will see how install and use a publication chain based on these three softwares.

2. Installation

In this chapter we will install the softwares composing the publication chain.

2.1. Install publican and poedit

publican is available into **Ubuntu** repositories. The installation of **publican** can be done with the following command:

```
sudo apt-get install publican libservlet2.4-java
```

poedit is also available into the repository and can be installed with the command:

```
sudo apt-get install poedit
```

2.2. Install serna-free

serna-freeis not available in the Ubuntu repository and not published as a deb package. We will then use the rpm package and the software **alien** to install it cleanly into our system.

```
sudo apt-get install alien
```

Then download the rpm from the official repository:

```
wget http://downloads.sourceforge.net/project/sernafree.mirror/ser
na-free-4.4-4.4.0-20111103.i686.rpm -O serna-free.rpm
```

Convert the rpm into deb using **alien** (It could take some time)

```
sudo alien -dck serna-free.rpm
```

Install the created package using the following command:

```
sudo dpkg -i serna-free*.deb
```

Execute the post-installation script to complete the installation:

```
sudo sh -c 'export SERNA_EXE=serna.bin; export SERNA_TAG=serna-free-4.4; export
 INSTALL_PREFIX=/opt; /opt/serna-free-4.4/bin/serna-postin.sh'
```

Create a link to the program into **`/usr/local/bin/`** to have it available in the path

```
cd /usr/local/bin
sudo ln -s /opt/bin/serna
```

Finally, add the shortcut into the desktop menu by creating the file **`~/.local/share/applications/serna.desktop`** with the following content:

```
[Desktop Entry]
Name=Serna
Comment=XML WISWIYM Editor
Exec=serna
```

```
Icon=/opt/serna-free-4.4/icons/SernaIcon32.png
Terminal=false
Type=Application
Categories=Application;Office;
StartupNotify=true
```

serna-free application is now available into **Ubuntu** → **Office** → **Serna**

3. Redaction

Now, the publication chain is ready then come the time to write a article or an book.

3.1. Creating the article's skeleton

The first action will be to create the skeleton of the document. We will take the redaction of this article as an example.

First we will ask **publican** to create the skeleton of an article with the commands:

```
mkdir ~/Documentation
cd ~/Documentation
publican create --type=article --name "Publican and Serna" --product "Publication Chain"
```

--type define the type of the document. It can be *article* or *book*.

--name define the name of the document.

--product define the name of the product the document is written for.

The command creates the a directory **~/Documentation/Publican_and_Serna/** directly constructed from the value of the parameter *--name*. This directory contains the **publican** configuration file **~/Documentation/Publican_and_Serna/publican.cfg** and five other files into the subdirectory **~/Documentation/Publican_and_Serna/en-US**/. *en-US* is the default language of **publican**.

Lets have a look into these files:

~/Documentation/Publican_and_Serna/en-US/Publican_and_Serna.xml

```
<?xml version='1.0' encoding='utf-8' ?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN" "http://www.oasis-open.org/
docbook/xml/4.5/docbookx.dtd" [
<!ENTITY % BOOK_ENTITIES SYSTEM "Template.ent">
%BOOK_ENTITIES;
]>
<article>
 <xi:include href="Article_Info.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
 <para>
  This is a test paragraph
 </para>
 <xi:include href="Revision_History.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
 <index />
</article>
```

The name of this file directly comes from the parameter *--name* of the command line. This is the root of the document. It includes other files generated automatically like the article information at the beginning and the revision history at the end, just before the index.

In this skeleton it is proposed to write directly the article inside this file. We will see in the next chapter how to make the document easier to edit by including sections into separate and dedicated files.

~/Documentation/Publican_and_Serna/en-US/Article_Info.xml

```
<?xml version='1.0' encoding='utf-8' ?>
<!DOCTYPE articleinfo PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN" "http://www.oasis-open.org/
docbook/xml/4.5/docbookx.dtd" [
<!ENTITY % BOOK_ENTITIES SYSTEM "Publican_and_Serna.ent">
%BOOK_ENTITIES;
]>
<articleinfo id="arti-Publication_Chain-Publican_and_Serna">
<title>Publican and Serna</title>
<subtitle>short description</subtitle>
<productname>Publication Chain</productname>
<productnumber>0.1</productnumber>
<edition>0</edition>
<pubsnumber>0</pubsnumber>
<abstract>
<para>
A short overview and summary of the book's subject and purpose, traditionally no more than
 one paragraph long. Note: the abstract will appear in the front matter of your book and will
 also be placed in the description field of the book's RPM spec file.
</para>
</abstract>
<corpauthor>
<inlinemediaobject>
<imageobject>
<imagedata fileref="Common_Content/images/title_logo.svg" format="SVG" />
</imageobject>
</inlinemediaobject>
</corpauthor>
<xi:include href="Common_Content/Legal_Notice.xml" xmlns:xi="http://www.w3.org/2001/
XInclude" />
<xi:include href="Author_Group.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
</articleinfo>
```

This file contains the information of the article: the title of the document, the product associated, the versions of product and document and an abstract explaining the subject of the article. It also includes other documents:

- The legal notice which is part of **publican** template.
- The list of author generated by the previous command.

~/Documentation/Publican_and_Serna/en-US/Author_Group.xml

```
<?xml version='1.0' encoding='utf-8' ?>
<!DOCTYPE authorgroup PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN" "http://www.oasis-open.org/
docbook/xml/4.5/docbookx.dtd" [
<!ENTITY % BOOK_ENTITIES SYSTEM "Publican_and_Serna.ent">
%BOOK_ENTITIES;
]>
<authorgroup>
<author>
<firstname>Dude</firstname>
<surname>McPants</surname>
<affiliation>
<orgname>Somewhere</orgname>
<orgdiv>Someone</orgdiv>
</affiliation>
<email>Dude.McPants@example.com</email>
</author>
```

```
</authorgroup>
```

This file gather the list of the writer of the document. The content of this file is included into the article info.

~/Documentation/Publican_and_Serna/en-US/Revision_History.xml

```
<?xml version='1.0' encoding='utf-8' ?>
<!DOCTYPE appendix PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN" "http://www.oasis-open.org/
docbook/xml/4.5/docbookx.dtd" [
<!ENTITY % BOOK_ENTITIES SYSTEM "Publican_and_Serna.ent">
%BOOK_ENTITIES;
]>
<appendix id="appe-Publican_and_Serna-Revision_History">
<title>Revision History</title>
<simpara>
<revhistory>
<revision>
<revnumber>0-0</revnumber>
<date>Fri Sep 21 2012</date>
<author>
<firstname>Dude</firstname>
<surname>McPants</surname>
<email>Dude.McPants@example.com</email>
</author>
<revdescription>
<simplelist>
<member>Initial creation of book by publican</member>
</simplelist>
</revdescription>
</revision>
</revhistory>
</simpara>
</appendix>
```

This file is gathering the list of revision of the article. It will be rendered as an appendix and will appear at the end of the document (since it has been included at the end of the root file).

~/Documentation/Publican_and_Serna/en-US/Publican_and_Serna.ent

```
<!ENTITY PRODUCT "Publication Chain">
<!ENTITY BOOKID "Publican and Serna">
<!ENTITY YEAR "2012">
<!ENTITY HOLDER "| You need to change the HOLDER entity in the en-US/Publican_and_Serna.ent
 file |">
```

This file is containing the definition of entities. This file is inserted every generated files. The values defined in this file can be used as parameter inside the XML document.

~/Documentation/Publican_and_Serna/publican.cfg

```
xml_lang: "en-US"
type: Article
brand: common
```

This is the configuration file used by **publican** during the rendering of the document and define the type of the document (Article) and the brand (common) which will define the look and feel of the final rendering. With **Ubuntu** package, only the '**common**' brand is available.

Lets now render the first article with the following command:

```
cd ~/Documentation/Publican_and_Serna
publican build --langs en-US --formats html-single
```

and see what has been produced in **firefox**

```
firefox tmp/en-US/html-single/index.html
```

We will see in detail how to use this command to generate user-friendly documentation.

3.2. Preparing the document structure

We have now to edit the files automatically generated and replace default value by real information. This should be done directly by editing the XML file into a text editor.

Edit **publican.cfg** and add the following lines:

```
docname: Publican_and_Serna
mainfile: Publican_and_Serna
```

This will allow to change the title of the document without changing the name of the XML files. If these values are not defined, **publican** will determine the name of the main file based on the title information written into the information file (**Article_Info.xml** or **Book_Info.xml**).

Edit the file **Article_Info.xml**, **Author_Group.xml**, **Revision_History.xml** and **Publican_and_Serna.ent**. Update their content with the information related to the project and authors.

Now to simplify the edition of our article, we will create sections and we will include these section inside the document root: **Publican_and_Serna.xml**.

Each section is stored into a file will contain the text and formating description of a chapter.

For the redaction of this article we created the following sections:

- **Introduction** stored into the file **Section-Introduction.xml**
- **Installation** stored into the file **Section-Installation.xml**
- **Redaction** stored into the file **Section-Redaction.xml**
- **Publication** stored into the file **Section-Publication.xml**

With the following content:

```
<?xml version='1.0' encoding='utf-8' ?>
<!DOCTYPE section PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN" "http://www.oasis-open.org/
docbook/xml/4.5/docbookx.dtd" [
<!ENTITY % BOOK_ENTITIES SYSTEM "Publican_and_Serna.ent">
%BOOK_ENTITIES;
]>
<section>
 <title/>
 <para/>
</section>
```

This XML code is describing a section. This section is containing one empty title and one empty paragraph.

We then updated the main document **Publican_and_Serna.xml** to include the section in the order we would like to see.

The root document have now the following content:

```
<?xml version='1.0' encoding='utf-8' ?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN" "http://www.oasis-open.org/
docbook/xml/4.5/docbookx.dtd" [
<!ENTITY % BOOK_ENTITIES SYSTEM "Publican_and_Serna.ent">
%BOOK_ENTITIES;
]>
<article>
 <xi:include href="Article_Info.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
<xi:include href="Section-Introduction.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
<xi:include href="Section-Installation.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
<xi:include href="Section-Redaction.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
<xi:include href="Section-Publication.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
<xi:include href="Section-Revision_History.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
 <!--index /-->
</article>
```

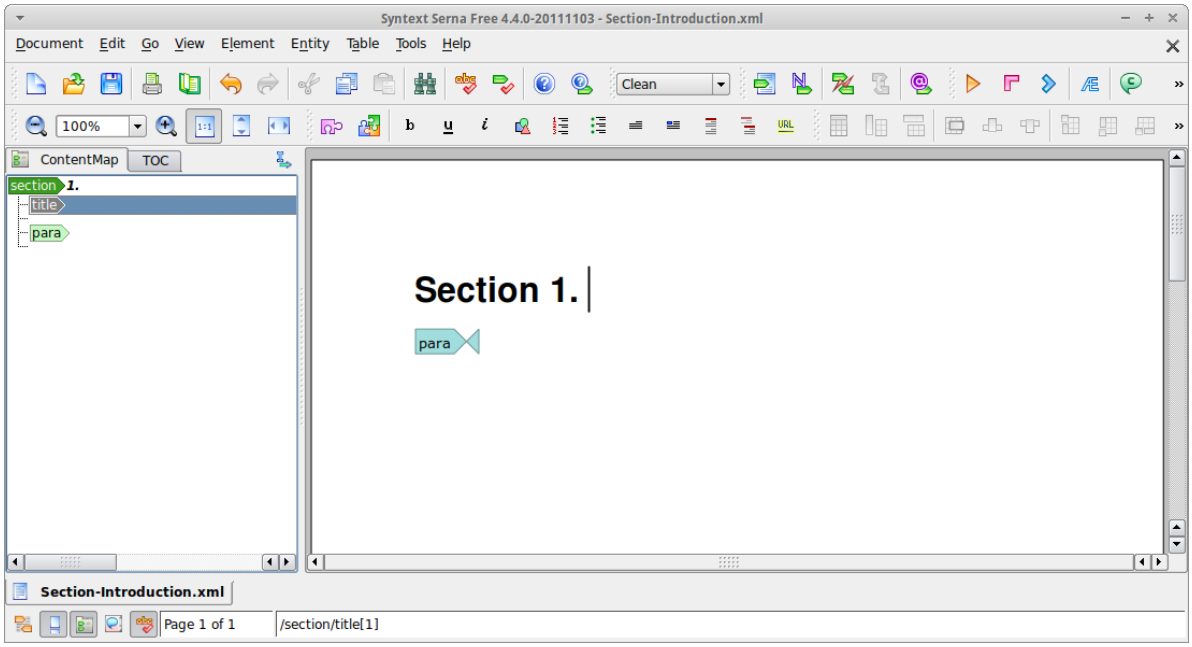
index has been commented out the since it is not needed for this project.

3.3. Writing the article

3.3.1. Using `serna-free`

The structure of our document is ready. We can start writing the article. To make this easier, we will use **serna-free**. This tool allow to edit the document in a **WYSIWYM** way. It also does the validation of the conformity of the document with the **DocBook** rules in real time. If you try to create a structure not supported by the **DocBook** definition you will see an error and the operation will be canceled..

Start **serna-free** an open the document **Section-Introduction.xml** using the menu **Document → Open...**



serna-free is divided into 2 main area: The *Content Map / TOC* on the left side and the edition area in the right side.

The *Content Map / TOC* panel displays a view of the structure of the document. This is an interactive area that allow to reorganize the data by a simple drag and drop.

The right side is displaying the text into a formatted view. This representation is not reflecting the exact view of the final document but is giving a quick overview of how it will look.

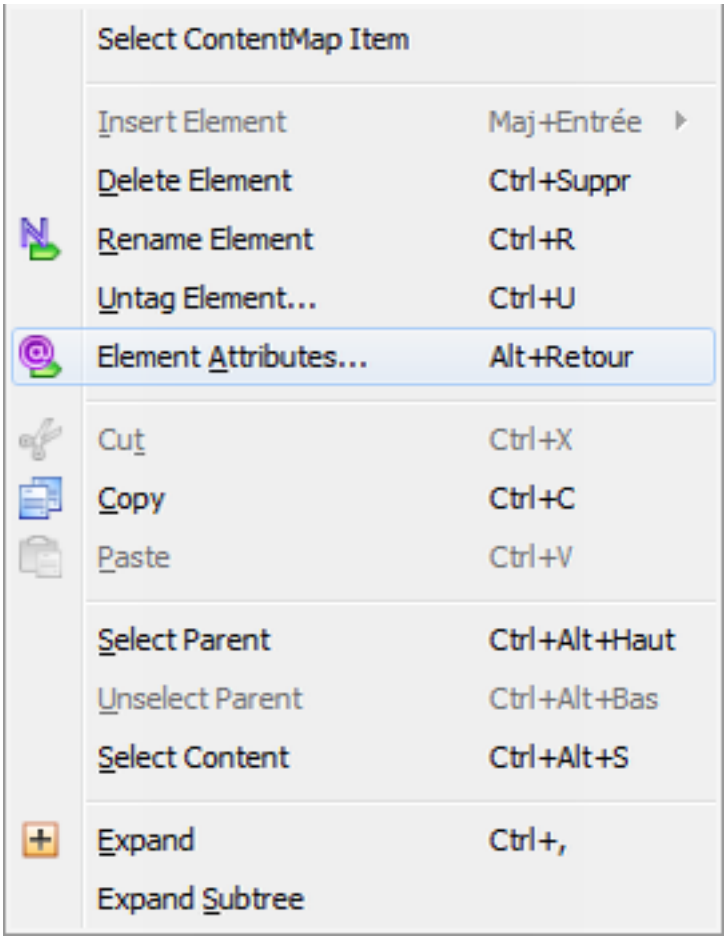
Using these two panels will allow the edition of the document. The first thing to do is the add the title of the section then the text of the paragraph.

In a **DocBook** article as well as into any XML document the formating is not done during the redaction of the article. We do perform a description of the formating using XML elements. This is the subject of the next chapter.

3.3.2. Docbook XML elements

This chapter is gathering some basic of **DocBook** element and explaining how to use them into **serna-free** and how it is encoded into **DocBook** XML.

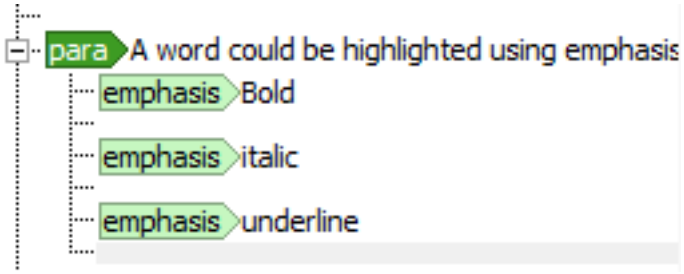
In **serna-free**, the contextual menu can be activated on both right and left panels. It will gives you the access to every possibilities given during the edition of your document. When a text is selected, it is possible to 'tag' the selected text and insert it into an XML element.



Highlighting words

A word could be highlighted using emphasis: **Bold**, *italic*, underline

To do so, select the text to highlight into the editing pane and click on the button of the tool-bar. The document will be modified as in the screen-shot below (Bold in bold, Italic in italic and underline underlined):



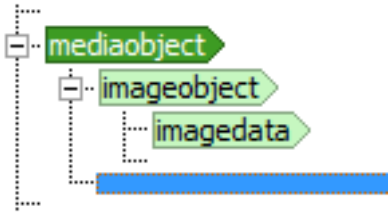
This is a graphical view of the underlaying XML which is in reality:

```
<para>A word could be highlighted using enphsis: <emphasis role="bold">Bold</
emphasis>, <emphasis role="italic">italic</emphasis>, <emphasis role="underline">underline</
emphasis></para>
```

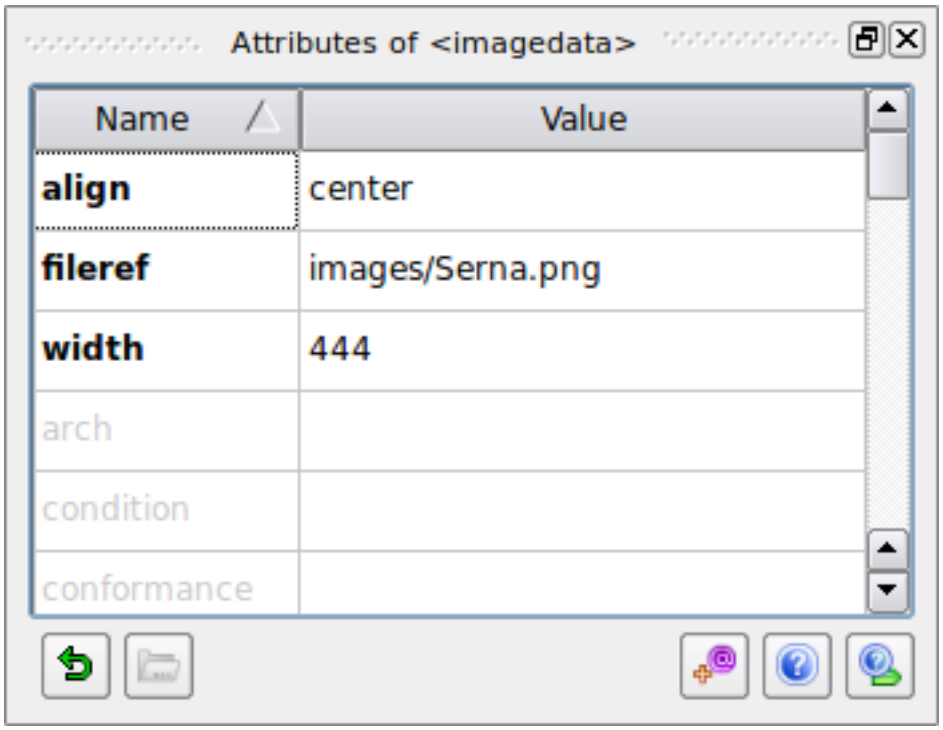
A word could also represent a **`filename`**, an **application**, a **`command`**, a **`code`**, a phrase, a **`class-name`**... There are a lot of possibility offered by the **DocBook** reference. To highlight a specific word or phrase, you should select it and from the contextual menu select **Wrap Info Element**

Images

Image are specific object and have to be inserted as **<mediaobject>** or **<inlinemediaobject>** composed by an **<imageobject>** composed by and **<imagedata>**.



The **<imagedata>** element contains attributes visible using the context menu **Element Attributes...**



The image to display is defined by the attribute `fileref`. You can use the little yellow folder in the bottom left to browse the image to insert.

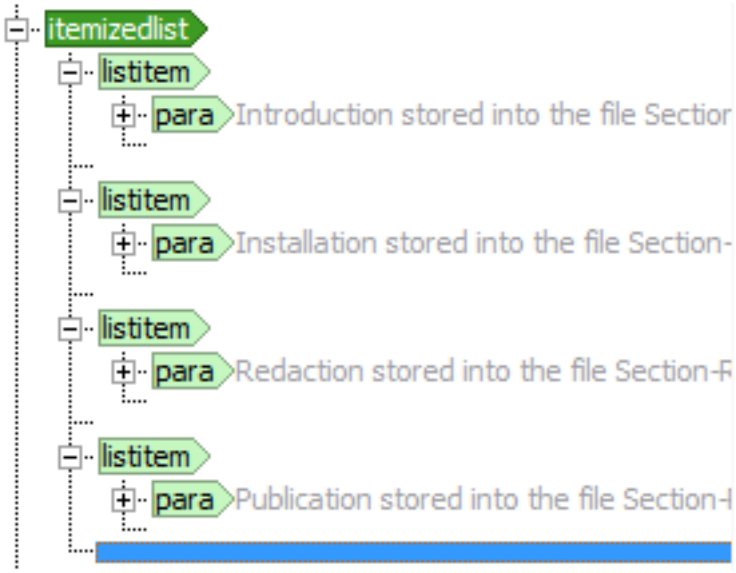
The underlying XML code looks like that:

```
      <mediaobject>
        <imageobject>
          <imagedata align="center" fileref="images/MediaObject.png"/>
        </imageobject>
      </mediaobject>
```

Lists

Two kinds of lists exist: **<itemizedlist>** and **<orderedlist>**. In the first list, each item is preceded by a bullet. In the ordered list, each item is preceded by a number (incremental).

The elements **<itemizedlist>** and **<orderedlist>** represent the entry point of the list. These elements are composed by a list of **<listitem>** containing the data to display a **<para>** like in the example bellow:



The underlying XML code looks like that:

```
    <itemizedlist>
      <listitem>
        <para>Introduction stored into the file <filename>Section-Introduction.xml</
filename></para>
      </listitem>
      <listitem>
        <para>Installation stored into the file <filename>Section-Installation.xml</
filename></para>
      </listitem>
      <listitem>
        <para>Redaction stored into the file <filename>Section-Redaction.xml</filename></
para>
      </listitem>
      <listitem>
        <para>Publication stored into the file <filename>Section-Publication.xml</filename></
para>
      </listitem>
    </itemizedlist>
```

In this document we only used **<itemizedlist>**. Creating a sub-listing can be done by inserting an **<itemizedlist>** or **<orderedlist>** inside a **<listitem>**.

Program listing

The element **<programlisting>** allow to insert pieces of code inside a document. **publican** as the capability to highlight the code to make it easier to read. To perform the correct highlighting, it is mandatory to define the language of the code. This language will be defined into the attribute language of the element. The list of supported language is available into the following page:

- *http://search.cpan.org/~szabgab/Syntax-Highlight-Engine-Kate-0.06/lib/Syntax/Highlight/Engine/Kate.pm#PLUGINS*

The first column list represent the text to add into the attribute.

To go further

Video how to use **serna** are available at

- *http://vimeo.com/groups/111908*

Information about how to write a clean **docbook** article are available at

- *https://hudson.jboss.org/hudson/job/PressGang_Documentation_Guide/lastSuccessfulBuild/artifact/target/docbook/publish/en-US/html/sg-Structure_Guidelines.html*

4. Publication

4.1. Rendering the document

Our document is now written in XML **DocBook** format but it is not so user friendly. We need then to publish it into another format. **publican** is proposing various formats:

- *`html`*: html document split in pages with navigation buttons on top and bottom of each page
- *`html-single`*: html document in one page
- *`html-desktop`*: The same as *`html`* but with an additional table of content in the left
- *`pdf`*: the document is rendered into one pdf file
- *`epub`*: the document is rendered as an ebook

The following command will generate a pdf document

```
cd ~/Document/Publican_and_Serna publican build --lang en-US --format pdf
```

`--lang` parameter is mandatory and can be the language code or the keyword **`all`**.

`--format` parameter defines the output format and should be one of the format previously listed. It is possible to generate multiple format output by separating the format by a comma.

The rendered document will be stored into **`./en-US/tmp`** directory. When the document is ready to be published the parameter *`--publish`* should be added to the command. It will store the document into the directory **`./publish/`*`Lang/ProductName/ProdVersion/Fromat/DocumentName/`** like for example: **`./publish/en-US/Publication_Chain/1.0/pdf/Publican_and_Serna/`**. These files will be automatically pushed to the web site if you are creating a documentation web site as described into the next chapter.

4.2. Managing a web site

In this chapter, we will see how to create a website that can gather the documentation and give an access to all the consecutive version of your document with a minimum of effort.

4.2.1. Creating the web page structure

Let's first create the web site using the **publican** command:

```
mkdir ~/Documentation/WebSite
cd ~/Documentation/WebSite
publican create_site --site_config website.cfg --db_file website.db --toc_path html
```

It create numerous files into the current directory:

- **`website.cfg`** gathering the configuration of the web site
- **`website.db`** gathering the information related to the document installed into the web site
- **`html/`** gathering the pages of the web site and the articles and books published

It is possible to override the style of the web site by adding formating into the file **`html/site_overrides.css`**. We will just create this file without modification to remove the possible error into the html code generated:

```
touch html/site_overrides.css
```

Edit **`website.cfg`** and append following lines

```
def_lang: "en-US"
search: '<div/>'
manual_toc_update: 1
```

In this example we do disable the search capability and we defined to update table if content manually. This is useful if you want to modify the table of content but new book will not appear into the table of content before you execute the command:

```
publican update_site --site_config ~/Documentation/WebSite/website.cfg
```

Be aware the this command will overwrite the **`toc.html`** files and will erase all the modification you might do in the file previously.

4.2.2. Creating the home page

We have the structure of the web site ready, we need now the create the welcome page that will be displayed to the users when they will reach the web site. The page is created as an article containing only one page; the welcome page.

Execute the following command the create the home page article:

```
cd ~/Documentation/WebSite/
publican create --type Article --name Home_Page
cd Home_Page
```

Append in **`publican.cfg`** the type of article we are editing:

```
web_type: home
```

Edit the file **Home_Page.xml** which is representing our welcome page:

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- This document was created with Syntext Serna Free. --><!DOCTYPE article PUBLIC "-//
OASIS//DTD DocBook XML V4.5//EN" "http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd" [
<!ENTITY % BOOK_ENTITIES SYSTEM "Home_Page.ent">
%BOOK_ENTITIES;
]>
<article>
  <title>What&apos;s up Doc!</title>
  <mediaobject>
    <imageobject>
      <imagedata fileref="images/title_logo.svg" align="center"/>
    </imageobject>
  </mediaobject>
  <para>The web site is not a blog, it is not a standard web site but it is a web site
 gathering various documentation I wrote around Linux. I write these documentation to
 remember how to do the thing with Linux and while I doing technical researches.</para>
</article>
```

In this example, we just add an image and a little welcome text.

The page is now ready to be published into the website.

Execute the following command to do so:

```
publican build --publish --formats html-single --embedtoc --langs all
publican install_book --site_config ~/Documentation/website.cfg --lang all
```

--embedtoc tells **publican** to add the table of content of the web site in the left of the page

To add a logo into the top left corner, you have to create the following image: **~/Documentation/WebSite/Home_Page/en-US/images/web_logo.png** and publish the modification of the home page with the previous command.

4.2.3. Publication of an article

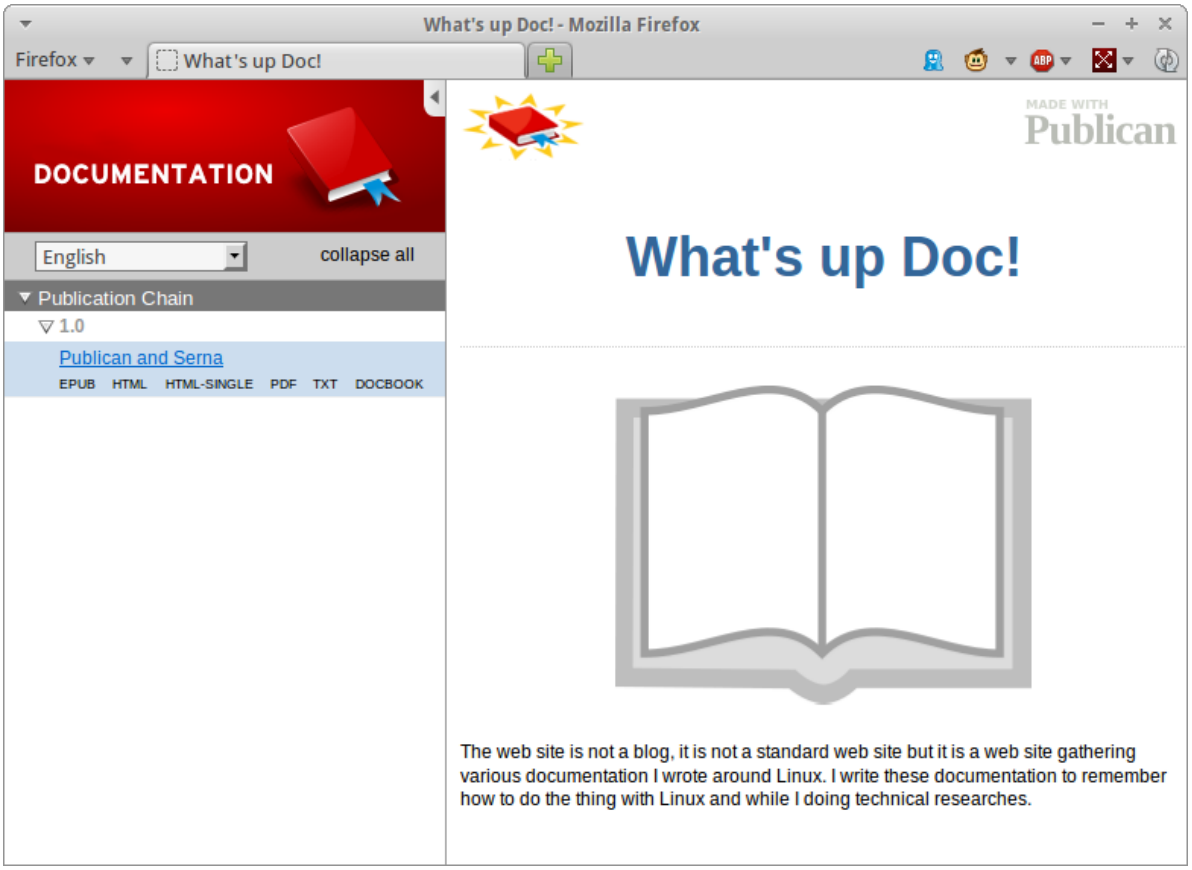
The publication of an article or a book in to the website is done with the commands:

```
cd ~/Documentation/Publican_and_Serna/
publican build --formats=html,html-single,txt,pdf,epub --langs=en-US --embedtoc --publish
publican install_book --site_config ~/Documentation/WebSite/website.cfg --lang en-US
```

Removing an article can be done by executing the following commands:

```
cd ~/Documentation/Publican_and_Serna/
publican remove_book --site_config ~/Documentation/WebSite/website.cfg --lang en-US
```

Here is how the web site will look like



In this screen shot, the files **`toc.html`** have been update to make the *TXT* and *DOCBOOK* output format available.

5. Translation

Our document and web site are now available in English. It could be nice to have them in other languages. We will see in this chapter how easy it is to translate the documentation we created with our publication chain.

When the document is frozen you can prepare the document for translation. This preparation will consist into the extraction of all the strings of the document into .pot file.

This is done with the command:

```
cd ~/Documentation/Publican_and_Serna
publican update_pot
```

Then you should choose the language you would like the add and execute the commands:

```
cd ~/Documentation/Publican_and_Serna
publican update_po --lang=fr-FR
```

In this example we decided to create the French version of our document. This command create the **`.po`** file in the subdirectory fr-FR. The **`.po`** files are the file we will use for the translation.

The command **`publican`** with parameters *`update_pot`* and *`update_po`* have to be executed when the original document has been modified. These commands will update the strings contained into the translation file.

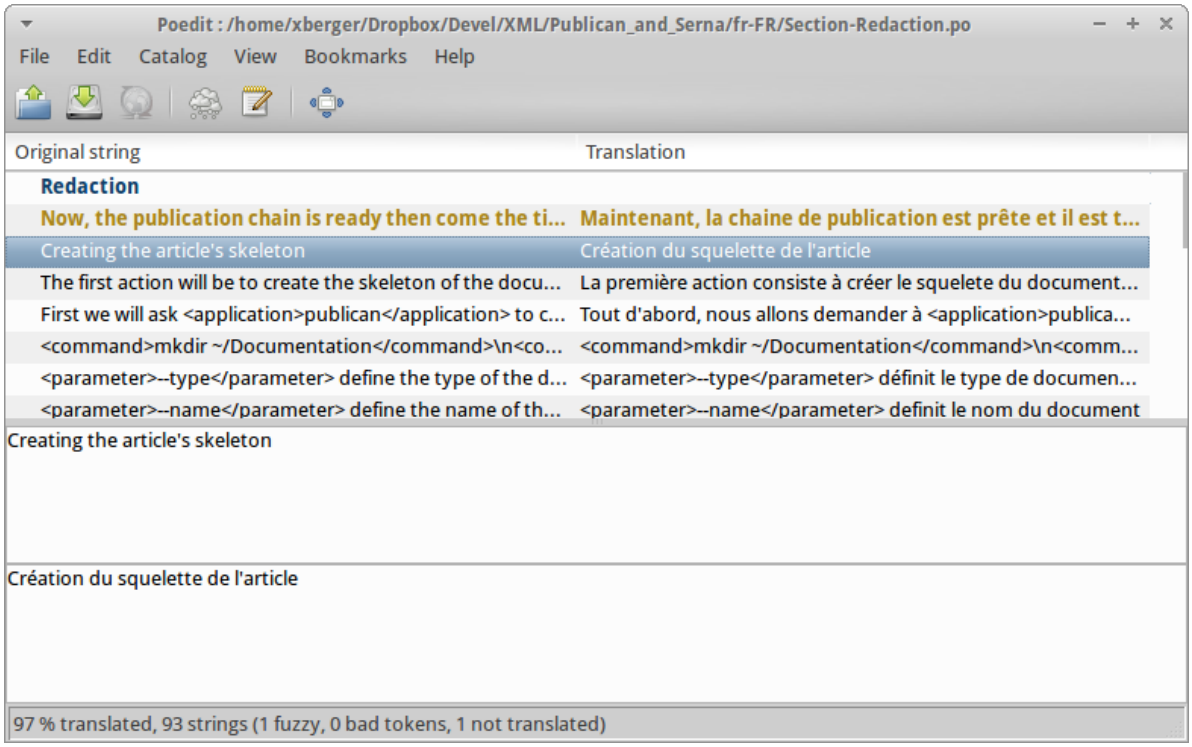
A **.po** file contains all the strings of a defined file and their translation. If you open a **.po** file you will see the following kind of lines:

```
#. Tag: para
#, no-c-format
msgid "<emphasis role=\"bold\">Publication chain</emphasis>"
msgstr "<emphasis role=\"bold\">Une chaine de publication</emphasis>"
```

The commented lines describe the tag and the status of the translation. *msgid* is the original string and *msgstr* is the translation.

It is possible to edit directly this kind of file but it is existing application that could make the job easier.

poedit is this kind of application. Start the application with the menu **Ubuntu** → **Development** → **Poedit**. After configuring the preferences of the application, you can open a **.po** file with the menu **File** → **open**. It the example below we did open the file ~/Documentation/Publican_and_Serna/fr-FR/Section-Recaction.po and we almost completed the translation:



The strings needing translation are highlighted in blue. The strings in yellow are flagged as fuzzy. This means that the source string has been changed since the last translation and that a review of the translation is required. This occurs when the **.pot** and **.po** file are update after a modification of the original document.

The two bottom panels display the original string and allow the translator to translate the sentence.

To create a new language it is also required to translate the welcome page of the website using procedure we just described.

The publication of a new translation is done like for a standard publication but with another language code with the commands:

```
cd ~/Documentation/Publican_and_Serna/
publican build --formats=html,html-single,txt,pdf,epub --langs=fr-FR --embedtoc --publish
```

```
publican install_book --site_config ~/Documentation/WebSite/website.cfg --lang en-FR
```

and, for the web site:

```
cd ~/Documentation/WebSite/Home_Page/
publican build --formats=html-single --langs=fr-FR --embedtoc --publish
publican install_book --site_config ~/Documentation/WebSite/website.cfg --lang en-FR
```

Then you will need to update the table of content with the command below after having saved the current **`toc.html`** that will be overwritten:

```
publican update_site --site_config ~/Documentation/WebSite/website.cfg
```

This will conclude this article about installation and usage of a publication chain. To go further in the usage of publican, refer to the official documentation available at:

- *http://jfearn.fedorapeople.org/en-US/Publican/*

A. Revision History

Revision 1.0 **Fri Sep 21 2012** **Xavier Berger** ***berger.xavier@gmail.com***
Ed1

Initial release of article

Note: Document improvement possibilities

- **Branding** customization
- **svn** to manage the document over the time
- **Sigil** to edit epub could be a complement and may be reviewed
- Add publication **scripts** to help publican management