LibreOffice & VML plain source text working and comparing and AsciiDoc

Dr. Hartmut Schorrig
www.vishia.org

2024-07-10

LibreOffice odt content is held parallel and also editable and convertible in a plain text, the Format is named VML (Vishia Markup Language). Also working with Asciidoc is supported.

Table of Contents

1 Approaches

Table of Contents

1.1 LibreOffice beside the plain text of content

LibreOffice and Asciidoc are two very different approaches to write (technical) documentation. Both have advantages and disadvantages.

One intention to use Asciidoc and LibreOffice parallel for the same document is: LibreOffice has the disadvantage that *“what you see is* ***what you have****”* is not true. It follows the known approach *“What you see is what you get”*, but some stuff is hidden which should be more obviously — The advantage of Asciidoc is: You see what you have. For example specific formats (styles) with its names, exact written relative link, etc. Asciidoc is a source format, it is a plain text without hidden stuff.

But Asciidoc has unfortunately a specific 'grown' syntax and cannot present all necessities of a well documentation. Therefore, a slightly different way was gone: Asciidoc is not used, instead a specific here defined markup format is used as counterpart for LibreOffice presentation. Asciidoc is generated ready to use for HTML output generation too. But editing in the plain text should be done with the specific VML markup format, (*Vishia Markup Language*) presented here.

The substantial approach for using LibreOffice is: It is proper for page oriented documents. Such documents can show technical things in a standard-two-page view on the current familiar large monitors, inclusively well positioned figures as explanation. It is better than the linear scrolling html view. But both may be necessary, a documentation should be available in both formats.

Last not least, editing the documentation in both formats, markup and in LibreOffice is an advantage. For *“What you see is what you* ***have****”* - content use the plain text markup, for appearance of the pdf document view use LibreOffice. Plain text markup format has also the advantage of comparability to older versions.

This tool converts `LibreOffice.odt` files to `Asciidoc.adoc` and the plain text markup, and back again from the `PlainText.vml.adoc` to `LibreOffice.odt`.

A side effect is: On back generation of LibreOffice from the plain text, a lot of junk which may be grown in the XML data is removed. This junk comes internally from used and remove again direct formatted text parts. But in conclusion, all exclusive some special direct formatting information are removed. They are not supported and not desired. See chapter 1.3 Using only indirect styles.

1.2 Why another markup format instead and beside Asciidoc?

The basically idea is, that LibreOffice is supplemented with a Markup language in plain textual form to see all ‘'*What you see is what you have*” with all internals. There are several markup formats, see https://en.wikipedia.org/wiki/-Markup_language or also in German: https://de.wikipedia.org/wiki/Auszeichnungssprache. There are some considerations to the markup language:

- Procedural markup: The markup contains statements how to print or render. Latex is for example partially a procedural markup. The principle is: Say what to do with the following text. For example Take an italic font with given name, then continue rendering.
- Descriptive markup: The markup describes the properties of parts of the text near the text itself or including the text. If the properties can be semantically oriented. It means, a text part is marked as “Quotation” because it is a quotation. Using an italic font is controlled by the style. The most known descriptive markup is **HTML** (*Hyper Text Markup Language*). The styles are placed in the associated CSS script (Cascading Style Sheet).

Also some proven markup languages exist. Latex should be known, also MD (*MarkDown* https://en.wikipedia.org/wiki/Markdown*)*.

Asciidoc is frequently used for software documentation, also because of the advantage, that Asciidoc can immediately include code snippets from sources in the documentation. But exactly this is solved a little bit abbreviating, see chapter Error: Reference source not found Error: Reference source not found page Error: Reference source not found. But nevertheless Asciidoc is selected firstly, also because Asciidoc was and is frequently used by me beside LibreOffice in the past.

LibreOffice is internally also stored as markup language, it is named “representational markup”. It means outside the user see the presentation (*wysiwyg*), inside all data are contained similar as a descriptive markup. This are the internal structure definition in the xml files `content.xml` and `style.xml` inside an `libreOffice.odt` file.

Asciidoc is by itself a little bit confuse in selecting formatting text. That’s why some discussions and also adaptions are made here.

To get a proper plain text editable markup format, which follows the structure of LibreOffice or a really proper text system, and it is also simple and widely compatible to Asciidoc, an own markup system, named VML (Vishia Markup Language) is developed and used. See chapter 3 Vishia Mark up Language on page 16

1.3 Using only indirect styles

Writing a document with any office tool, you are inclined to use the simple direct formatting possibilities. Make a paragraph a little bit lesser meaningful, oh use italic font with a little bit lesser size, looks nice. All the office tools supports both, indirect and direct formatting. What is indirect formatting: Using style sheets. With style sheets, you can associate to any paragraph or text area a meaning. Not immediately the output format. The style sheets give the text also a semantic. A `Quotation` as style markes a text first as quotation, not *“print italic”*. Then you can set proper outfits to all used styles, and the whole text is proper changed if the necessary outfit should be changed.

If you want to express an additional info, lesser meaningful, you can use a style `AddInfo`. Or if you want to express a snippet from code, you can use a style `CodeJava` or on another part `CodeCpp` instead assign a monospace font with any size and maybe a back color. It is similar a semantic label to this part of text, it defines what it is.

Then you can set the style in the *Styles* side bar, define it, or use it from another document, and give it a proper outfit, the desired font, appearance etc. Then the appearance of that text parts are all equal in the text, and that is it, what is necessary.

Using indirect styles is a very old and proven technology. It is present in Word for DOS since the 1990^th^, present of course also in LibreOffice since beginning, present for HTML in the CSS style sheets, and such text writer systems as Latex and also Asciidoc works exclusively with indirect styles.

You can see the very familiar *italic* or *bold* also as indirect styles with this names, or also translate it to the indirect character styles `Quotation` and `Emphasis` which has a semantic meaning.

Direct styles complicates a document. All office tools support direct styles, because there are a lot of users, which do not write important documents, they write for there own and does not know the indirect style usage and advantage. But all professional document writer should only use indirect styles.

In (all) Office tools because of too much direct styles which were used, then deleted etc. there is a lot of data in the document which are meanwhile nonsense. On translation from LibreOffice to the plain markup text VMU and also Asciidoc this nonsense direct style entries but also all other direct formatting styles are ignored, except a few important ones. Translated back to LibreOffice this direct styles are removed. It is a cleanup process which may be important on dealing with large documents.

1.4 What you see is what you have

This is a very important saying, but not in all brain. The all known “*What you see is what you get”* instead is very known and it hides the view to “***what you have”***.

For example, links in the document. What you see in the text is: The text to the link, not the link itself. If you open the link (*Insert - Hyperlink*), then you do not see the real used link in LibreOffice, you see the absolute file position. (It is a RFC, Request for change, internally in LibreOffice Bug 128216 to see also the relative path.

If you want to change some more relative paths in its start point, because your directory tree is a little bit changed, then in the plain text source file `Plaintext.vml.adoc` you can relative simple use *search and replace* to gather and change all. In LibreOffice you must painstaking open each link, with the mouse, think what is happen etc. pp. And if you have your result-pdf, you may get bad surprises — LibreOffice may work exact. But you may make some mistakes while the painstaking work, then do it again.

Just holding the sources in both forms, as a `LibreOffice.odt` and also as `Plaintext.vml.adoc` with the same content, you can do the work where it can be done better. Improve your page layout and format text content in LibreOffice, and correct links, sections, Overview over chapters, images in the `Plaintext.vml.adoc`. After finish work in one file, you should only start the conversion to the other one, which needs about one second.

1.5 Working in the document in LibreOffice and similar in VML

The advantage of both tools can only be used if you can work in both for editing.

For that, a converter is provided, which converts either `LibreOffice.odt` File format to `Plaintext.vml.adoc` and vice versa. Then after converting you have both, can look and further work with both. But if you edit one of them, you should convert to the other format to use both and can edit furthermore with both.

In `Plaintext.vml.adoc` you have the advantage to compare simply the files, to see what is changed. In that manner also an editing by mistake of an older version can be fixed. But you should look anyway to have both file formats in the currently version.

Of course, if you can use both editing approaches, and converting the documents, you can only use an intersection set of capabilities of both formats. But this intersection set has enough capabilities.

This intersection set is discussed in chapter Error: Reference source not found Error: Reference source not found.

2 Some decisions how to write a technical documentation

Table of Contents

2.1 Writing style in columns for each (sub) chapter

The first what should be obvious for this document is: It is written in columns. Reading in columns has the advantage, that the eye of the reader can capture the text in a vertical movement. Because the lines are not too long. You can fast capture the content, for example while searching a catchword. This style was familiar in the old years of printed documents, for example in encyclopedias. but it was forgotten in a time of html browser for first small screens. Now here the idea is recovered.

And a second advantage is: The column width is also proper for reading a pdf on a smart phone.

But in difference to the familiar column style in news papers, the columns does not go over the whole page from top to down, they are regular broken on each new chapter title, and also on an image or figure which needs the page width.

LibreOffice does support writing in columns, but for editing some times a little bit difficult to handle. But editing parallel in `vml.adoc` it is a little bit more supported.

2.2 Manual column or page breaks and positions of images

2.2.1 Page breaks and reserve space on page end

The reason or intention for page breaks is: You want to present a closed content on two pages side by side. Whereas in book view mode or for the printed document the left page has the even number and the right page has the odd one. This is important. Normally pdf viewer should be able to set for this mode.

This means also, a new chapter should start on top of an even page to have two pages side by side for the overview. But in opposite of this rule, it is often recommended to start a new main chapter on right side in a book, the same side as the title was written. This is a little bit contradictory. Never you should start an important new chapter near the end of the right (odd) page, that is stupid. Means, insert manually a page break before.

Using page breaks on proper positions in the text helps also that the page disposition is not sensitive to confuse on each small text changes. You have some space before page breaks.

2.2.2 How to insert a page or column break

Traditionally it seems to be proper to insert manually a page break with Menu “Insert – Page break” or “Insert – More Breaks”, and then select “Column break”. But effectively, the property of breaking the page is a property of the paragraph format. It means following the idea of direct formatting, to to the “Format – Paragraph”, then select the tab “Text flow”, then check the “Insert” box with Type: “Page” or “Column” and use Position: “Before”, that is proper. Then you get a column or page break before this paragraph. And this is usual what you want. The same is done internally if you use the Menu “Insert – Page break”, but the paragraph is split on the cursor position, some times unexpected. Using the “Format – Paragraph” is more simple.

In the plain text VML presentation the page or column break is written in a line before the paragraph as

```
<:pageBreak.>
```

This forces on back conversion to LibreOffice exact the above described behavior, a paragraph style with Insert Page break or column break before. It is compatible and obviously.

To simplify a page break in the current text, you should use the style `TextPg` or `TextCol` instead `Text`. This is also done automatically by back conversion.

2.2.3 Position of images

Traditional often images are positioned as possible, depending of the page formatting. For example Latex has its own free style to positioning images on a proper position in meaning of the Latex rendering, not in meaning of the user.

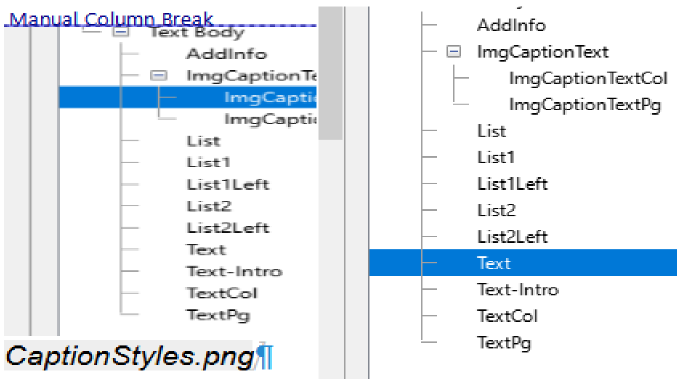
But familiar in html (often used for technical documentation), images are always exact positioned in the text flow. For technical documentation this may be important to have the images closed to its explanation.

That’s why this converting system between LibreOffice and the plain VML text, which has a natural closed relationship between text and image because of their sequence in the plain text presentation, uses a simple presentation of images in LibreOffice:

Images are always bounded to a paragraph which contains the image caption. The position is left or right bounded, but with 0 distance to the paragraph. If you move unintentionally the image in LibreOffice, go to its properties (right mouse) and entry 0.0 for its position, and the image should be proper again.

But there may be sometimes a problem: If the LibreOffice rendering for the page would insert a page break or column break at an inappropriate place near the image, then you should insert a manual page break or column break before the paragraph to which the image is bound.

The figure above is inserted with this shown style `ImgCaptionTextCol`. It forces a column break before, as also see in the image.


Manual Column Break
Text Body
AddInfo
ImgCaptionTe
ImgCapti
ImgCapti
List
List1
List1Left
List2
List2Left
Text
Text-Intro
TextCol
TextPg
CaptionStyles.png¶

AddInfo
ImgCaptionText
ImgCaptionTextCol
ImgCaptionTextPg
List
List1
List1Left
List2
List2Left
Text
Text-Intro
TextCol
TextPg

Figure 1: ImageCaptionStyles.png

But you should taken care about the new column, it should be more filled that the column before, elsewhere the rendering may fill the columns in an equal kind and inserts an unexpected new column here.

This handling should be done usual in the LibreOffice presentation. Then you see the image close to the text in the plain text VML presentation too. If you edit their, be careful with page breaks and column breaks, which are proper syntactically able to see in the plain text VML, and all is proper.

The numbering of the images is done by the VML to LibreOffice converting, not from LibreOffice itself. See chapter 3.7 Images page 19, because the `ImgCaptionText...` cannot support numbering.The behavior of tables is similar to that of images Make sure that the page and column breaks are proper.

LibreOffice has a little bit trouble if images are shifted with the mouse. This is a concession to the user who wants to have free mouse positioning, but exact this breaks the relationship between text and images. A second problem is handle image captions in a text box, with two sources of positioning errors.

2.3 Using a real small set of format styles and less direct formatting

Think about the proven rule “less is more”.

2.3.1 Is a free styled document design proper?

If you are concentrate to text writing with an office tool, you may be triggered to use a lot of nice styles for free possibilities for your design.

But also for Asciidoc, html, most Markup sources, the style of the currently text is only one. It has not a specific style dedication in the VML (and also in the other markup) languages), it’s only text. The converter from VML to LibreOffice takes the Paragraph style `Text` (not `Text Body`) This style should be used and defined in LibreOffice. The rule *“less is more”* produce a more relaxed design, concentrate to content, not to appearance.

2.3.2 List appearances

For Lists, usual the given list styles are used, which have the possibility to select different bullets etc. But especially the bullets can be also part of the text itself. This opens the opportunity to use a context related bullet, which can be also a specific text. Also for lists, they have the possibility of the “numbered list” with an auto incremented number in different styles. That is proper for common articles. But for exact presentation of technical things, the number should be related to the text, should not be automatically incremented and hence changed if a new list item is added. That’s why using a numbered list is not recommended in my mind. Write the bullet appearance by yourself.

All markup languages supports a numbered and an unnumbered list, also VML by using the familiar from Asciidoc known writing style

```
* list item
```

- list item using this appearance of the list style in LibreOffice as non numbered list. You can adapt it.

But it may be recommended not using a list style for lists, instead a specific paragraph style with the necessary indentation, and write the bullet manually. This styles can be defined with any specific user name in your document, but the recommended style names are:

● `List1`: The bullet point is manually set, it is contained in the standard UTF coded character set. You can copy it from this document here. The indentation here is 24 pt, but the first line has -18 pt (6 pt from left) in this document, it is able to adapt. Use a tab character after the bullet.

a) `List2`: And this is a manual written bullet which can be used in the further text as link. The deeper indented list has here 48 pt from left, and the first line -18 pt, which is 30 pt from left.

● `List1Left`: This is a list which is proper for shortage of space for the line in a column. The indentation is 0, but the first line is 6 pt and the tab is placed on 24 pt.

b) `List2Left:` But also this kind of List exists for the space saving writing style.

This here used list styles are paragraph styles, able to find short to all other paragraph styles. In VML there are written in form:

```
<:p:List1>•\t List1: The bullet TEST
```

The format style is given first in the paragraph line. The bullet is coded in UTF-8 in the text.

`\t` with the following space presents the tabulator character. The specific text style is written with `<:cStyle:any text.>`, but see next.

In this kind some more specific paragraph styles can be used. For using Asciidoc this is translated to:

```
[.List1]
• [cStyle]`List1`: The bullet point
```

The possibility of `[.List1]` in Asciidoc creates in HTML:

```
<div class="paragraph List1"><p>…</p></div>
```

It means it is presented by a so named division, which has a specific paragraph appearance controlled by the CSS script (Cascade Style Sheets for HTML).

2.3.3 Code snippets

As also able to see above, Code snippets are often used in a technical software documentation. The important feature is: The lines shouldn't be wrapped. For Asciidoc in HTML there is a nice feature, a sub window with the code with a horizontal slider. But this is never usable in a printed document where LibreOffice is source of, and it's also not possible in LibreOffice. The lines are broken on end of the paragraph width.

That's why the source text should be limited in text line width. For readability this is usual proper, because this code snippets are only snippets for illustration, and not the complete code. Read and edit the sources in the proper IDE (Integrated Development Environment) to work with it!

But an effort is necessary to copy source content to the documentation. Asciidoc supports that, it copies the content during HTML generation in the HTML. This feature is not usable for this LibreOffice approach, Instead there is a part of the translator from VML to LibreOffice which updates the code from the original sources, controlled (similar but better as in Asciidoc) by specific entries in the code. It means code snippets which are identical with the currently sources can be produced with less effort. This tool cuts the line to the given parameterized line width. So no unexpected line break is given.

But for that, of course, the code snippets should have, as often usual, a monospaced font. And also in the paragraph style the check box *“Do not add spaces netween paragraphs of the same style”* should be activated.

The code snippets are presented in VML as

```
<:Code:Adoc>
The original line in the source
```

<.Code>

In the Asciidoc format it is:

```
[Source, Adoc]
----
The original line in the source
----
```

Asciidoc interprets some stuff in the original line given here, it should not be contained (sophisticated). VML to LibreOffice regards Designations of specific characters introduced with the back slash as for example `\t` for a tabulator, and character styles, which are written as `<:style:content.>` A found `\t` in the code for example to output a tab character in a `printf("\t")` line in C language is replaced in VMU by `\\t` to prevent faulty interpretation. If you mark a code sequence for example with

```
A code line with // (1) in comment,
```

it is a marker proper for documentation, then the VML code contains:

```
<:Code:Java>
A code line with // (1) in comment,
```

<.Code>

The red `(1)` is the LibreOffice character style `cM` for "Marker", very short and concise. You can write exactly this in your source line, because (sensitive) it's part of the comment and it is understandable by a source code programmer which knows that concept. In Asciidoc this appears as

```
[Source, Java]
----
A code line with // [cM]`(1)` in comment,
----
```

Which Code paragraph styles should be given: This depends of your requirements. You can define it by your own. The default LibreOffice paragraph style `Code` should be used for common. But for specific languages some styles below Code in the Hierarchy should be given:

- `CodeCmd`: for common command lines
- `CodeScript` for common scripts. The designation is short.
- `CodeCpp` or `CodeJava` for programming languages
- `CodeVMU` and `CodeAdoc` especially for this document.

The font styles may be identically, or different in bold or italic, for your own. Usual the background color should be selected for recognizing the language. This is not so proper but acceptable for a white/gray/black printed document, but proper for a pdf viewer. Use a pastel color for the background.

2.3.4 Character styles

The usual used bold, italic, underlined, and also subscript and superscript character style dedications are all direct styles. If you press *ctrl+M* (*"Format – Clear Direct Formatting"*) which may be sometimes necessary, this designations are removed. Instead you should always use indirect character styles for that. All of this possibilities are given with the indirect styles.

But for compatibility and fast writing using the known hot keys as ctrl+I for Italic, the detected direct styles in a LibreOffice document are automatically translated to the necessary indirect style in the VMU plain text. While back conversion to LibreOffice you get the indirect formatting in LibreOffice for further working. You should know that, you have not effort, and you have the possibility to change the appearance for all marked texts in a unique kind.

The per default used character styles for the replacement of the direct styles are:

- `Quotation` (*Standard style in LibreOffice*) instead italic direct style, `<:Q:text.>` in VMU, `__text__` in Asciidoc.
- `Strong Emphasis` (**Standard style in LibreOffice**) instead bold direct style, `<:S:text.>` in VMU, `**text**` in Asciidoc.
- `Emphasis` (***Standard style in LibreOffice***) instead italic bold direct style. This is the standard appearance

of this style, `<:E:text.>` in VMU, `__**text**__` in Asciidoc.

- `Subscript` for $_{\text{Indices}}$ (user defined style in LibreOffice) instead subscript direct style, `<:1:text.>` in VMU, `~text~` in Asciidoc.
- `Superscript` for $^{\text{Indices}}$ (user defined style in LibreOffice) instead superscript direct style, `<:2:text.>` in VMU, `^text^` in Asciidoc.

For this character styles which should only influence this given properties of the text, and not the font size etc. LibreOffice works exact, but it hidden its exact working and it is error-sensible. What's happen: If you change the character style and you do not entry a new font or fount size, all is ok. The font and its size is derived from the paragraph style. But if you change the font for this character styles, it is changed. You can never revert it to "derived font". This is a problem of LibreOffice, should be fixed. But it is able to fix by manually handling of the internal `styles.xml` respectively replace a changed `styles.xml` by a proper one from another document, see 6.1 Exchange and maintain the styles of the document page 27

Additional you should have all code fonts and backgrounds which are existing as Code paragraph styles also as character styles. This allows refer to code snippets with the same writing style also in your currently texts. This styles should start all with "c", it is necessary for the VML conversion. After the "c" for some standard code styles only one character should be written. This is proper (but not necessary) for VML. The character styles in VML are written generally as `<:style:text.>`. For example it is proper readable and writable: `<:cC:float var; // C/++ reference.>` which appears in LibreOffice as `float var; // C/++ reference` or `<cM:(M).>` for a marker in a source which appears as `(M)`.

2.4 Character set and special characters

LibreOffice works with all available characters defined in the UTF character set. The VML file and its editor uses UTF-8. That is free.

But some characters especially which are not proper readable and writable in the VML file are replaced by transcriptions, see chapter 3.8 Transcription of specific characters page 22. Also storing the VML in Standard 7 bit ASCII is supported but not recommended.

2.5 Internal links, bookmarks

All chapter title should have proper mnemonic bookmarks. They are used for chapter references. The bookmarks or labels are written in the VML plain text in form

```
=== chapter title <:#chapterLabel.>
```

In Asciidoc there are used similar, the `chapterLabel` is generated also to the HTML document as anchor usable in the URL. In LibreOffice it is a bookmark.

References to internal bookmarks are written in VML in form `<:@ref:#ChapterLabel:3.8 Title>` but the Title and its number is not used for back translation to LibreOffice, it is automatically created there. It is an information in VMU after translation from LibreOffice.

Links to the file system should be used in LibreOffice generally as relative links. They are proper able to see and editable in VML, for example if the relations in the relative paths are a little bit tuned. This is one of the important advantage of VML. See chapter 3.x TODO.

2.5.1 External links to javadoc local files and the internet

This is a special approach for documentation of Java programs, but it can be applied similar also to other programming languages. The topic is: The software should be documented. Yea.

But how? There a three or four levels:

- The inner level is the software itself. Should lines be commented? Clear programming says “no”, users which studie old software says “yes please, why this statement …?”. The problem on immediately software documentation is: It is not maintained if the software is changed. That’s why some people speak about “clear programming” and require that the identifier, the names of variables, classes, operations are proper. That is right.

- In Java it’s familiar to have a documentation also in the source, for each element of a class (usual operations) and for the classes. This documentation should be, and is written in a common understandable style, and from this information in the source a usual HTML document is generated. This is Javadoc. It’s nice, it is familiar since many decades.

The Javadoc itself may be satisfying, is it? It is satisfying to explain usage of a sub class. It is not satisfying explaining a tool written in Java.

- The next level is, explain how the tool in Java works. How the software works. This may need graphical overview, maybe using UML tools, or at least an explanation about functions and operations from the user’s view. This is the minimum. And now it is nice to have to link from this explaining document to the Javadoc generated stuff, because both supplements each other. Javadoc contains the exact description of an operations, or of a class, all what this does or contains, but it does not explain how and why to use. For that the here written LibreOffice may be responsible to. See chapter 4 Implementation page 24

And now the request for the documentation:

First, it should be possible to work without internet. Yes! Presumed the Javadoc from your own sources is side by side to the document, or the Javadoc from a used source can be downloaded one time locally via zip. unzipped and places side by side to the downloaded pdf document. Then you can work without internet.

If you open a pdf document in the browser, not downloaded, it is also sometimes possible to open a relative link in the document without problems if the destination of this relative link is on the same location in the internet.

But sometimes, the relative linked destination, for Javadoc or other, is not available. That’s why I place a relative link and a link to the proper internet location one after another in form: See WriteOdt.main(...) (www). Both refer the same, local and in internet. The local link may contain the class and operation, as shown here, the www does not need its repetition.

But this is not the main problem. The problem is, that the link to an operation in Javadoc with a lot of arguments is sophisticated. And if you change only one argument type, or add one argument more, while software development, the link is faulty. You should search and adapt in your documentation.

This work can be done automatically and is done by the VML/LibreOffice translator.

As argument on compilation you need:

```
-www:https:myWeb/dir
```

In the VML you can write the link to the adequate operation as:

```
<:@link:./../docuSrcJava_vishiaLibreOffc/
org/vishia/odt/readOdt/WriteOdt.html#main::
WriteOdt.main(...).>
(<:@link:https://vishia.org/LibreOffc/...::
www.>
```

No more is necessary. The path to the local link should match till the name of the operation, do not need the correct label. The text to the operation does also contain only the operation name and the `(...)` with the ellipse `...` as placeholder.

Now the VML to LibreOffice translator reads the local existing Javadoc file, reads all anchors of operations, sort it by name and builds an index of the label (anchor in HTML) sorted by the name. If an operation with the same name, and hence different arguments is given, the first one is added to the index only. To refer to the second (in order in the HTML file), you need the full label.

If the first local relative linked operation is satisfied, the same path is used also for the www-link, starting from the `...` and starting in the relative path after the first `../`. That results in:

```
https://vishia.org/LibreOffc/
docuSrcJava_vishiaLibreOffc/org/vishia/odt/
readOdt/WriteOdt.html#main-
java.lang.String:A-
```

For back translation the same is done.

- First it is tested whether the text to a link in LibreOffice ends with `(...)`. If it is so, then it is (should be) the link to an operation with non specified arguments.
- Then in the link the hash # is searched, and after them the identifier. The link is shortened to this information only.
- And also, the part of the path after the `../` of the relative paths is stored to test the next coming www link.
- If the next coming www link starts with the given `www:https:myWeb/dir`, then the stored path is searched in the www link. If found then only that part till the equal path is stored, following with the `/...`. That restores the original given designation, or produces .

3 Vishia Mark up Language

Table of Contents

3.1 Basic Considerations

3.1.1 Plain source text

The important basic consideration is: It should be based on plain source text. The encoding should be UTF-8 to support also rarely letter also in LibreOffice without transliteration. But for exception situations (using an old editor) also US-ASCII or ISO-8859-x (8 bit width character coding) should be possible.

Some special non visible characters should be transliterated, see list in chapter 3.8 Transcription of specific characters page 22

There are only a few character sequences which controls the structure. Outside of paragraph texts there are more possibilities, see 3.1.3 Section, chapter and paragraph structure near Asciidoc Inside a consecutive text (in a paragraph) only `<:xxx.>` and the transliteration with `\x` is used. This prevents confusion with text parts as in Asciidoc, the known Asciidoc's `pass:[text]` for a non interpreted text and its also confusing abbreviation `+text+` is not necessary. See 3.1.4 Text structure similar Asciidoc but other designations page 17.

3.1.2 Comment lines

```
// Comment
<:Comment:marker>
  block comment
<.Comment:marker>
```

A line starting with `//` is ignored, but not inside a code block. This is also true inside lines of a paragraph. The comment line does not break the paragraph block.

The block comment can be used also to disable blocks of text. Nesting is allowed (TODO?).

3.1.3 Section, chapter and paragraph structure near Asciidoc

```
== chapter title Error: Reference source not
found Error: Reference source not found

<:p:style>
paragraph one line per sentence
or broken inside.

Next paragraph in standard style.
```

```
* A list
** With sub items

<:Section:style>

paragraph in section, maybe in columns.

=== sub chapter in section <:#label2>

<.Section>
```

Sections are parts of the document containing paragraphs and also complete chapters, which have a specific format. Especially this is used for writing in columns. Also have a specific background color for parts of the document is possible.

The chapter and paragraph structure is basically similar in Asciidoc, Mark Down, Wikipedia text. Here the basically chapter and paragraph structure of Asciidoc is used, with some specifics. It means:

3.1.4 Text structure similar Asciidoc but other designations

Inside a paragraph text and also all other texts (list items, chapter title etc) normal text is written as is. The UTF-8 coding allows using also rarely specific characters. Only specific character are transcribed, which are not able to show in normal text coding. That are for example the non breaking space, UTF-16 coding `\u00A0`, written as `\ ` in VML. See chapter 3.8 Transcription of specific characters page 22

All character designations uses character styles of LibreOffice. But there are some shortcuts for the standards, see chapter 3.9 Using Character styles, semantic text span at page 23. The general solution is writing in the text:

```
...text text1
```

There is only this one control sequence `<:  .>`. This is very more simple and obviously than the many specific designations in Asciidoc and some other markup language which can conflict with the normal text. Hence the known Asciidoc's `pass:[text]` for a non interpreted text (with special designation as to write) and its also confusing abbreviation `+text+` is not necessary. To write a <: and .> itself in the non styled text in VMU you should transcribe it with `<\:`. and `.\>`, for example to explain VMU itself. Also the \ can expressed by `\\`. No more is necessary.

3.2 Chapter designation and content

A chapter title line starts with

```
=== chapter title <:#chapterLabel>.
```

The number of `====` describes the deepness of the chapter. But other than in Asciidoc a label for the chapter is given on end in form `<:#label.>`.

A chapter can contain paragraphs, lists, code snippets, images, tables.

3.3 Writing style of paragraphs

A paragraph starts with a new line with an empty line before. All lines below which are not empty and do not start with * (for List items) are part of the paragraph. A line separator between is ignored. More as that: It is recommended to write each one sentence in a new line, and also a part of a sentence on a long sentence. The plain source text should not have the necessary of wrapping lines in editor.

A paragraphs can have a specific style. In Asciidoc this is able to express with [.style] before the paragraph, builds a <div class = style> in HTML. Instead in VMU it is designated also before the paragraph with <:p:style.>.

3.4 Lists

The items of a list starts after a * or more ** also without empty line between, but recommended write an empty line before each list item. The list itself is not specific dedicated. The items builds the list.

3.5 Code snippets

Code snippets starts with <:Code:style>. Following lines are presented each as one line, till <.Code>. The code lines are shown usual in a monospaced font after rendering (in LibreOffice). The lines are never broken. It is possible to include code snippets immediately from sources via link (recommended as relative link).

3.6 Sections

In LibreOffice unfortunately there is no indirect style for sections (2024-06). Hence some sensible section styles are defined in the VMU itself, as virtual (de facto, non formal) indirect style. Sections are enclosed by `<:Section:style>` and `<.Section>`.

3.7 Images

Images are always bounded to a paragraph which contains the image caption. The position is left or right bounded, but with 0 distance to the paragraph. If you move unintentionally the image in LibreOffice, go to its properties (right mouse) and entry 0.0 for its position, and the image should be proper again. On Translation VML to LibreOffice this positions are 0. See also 2.2 Manual column or page breaks and positions of images page 6

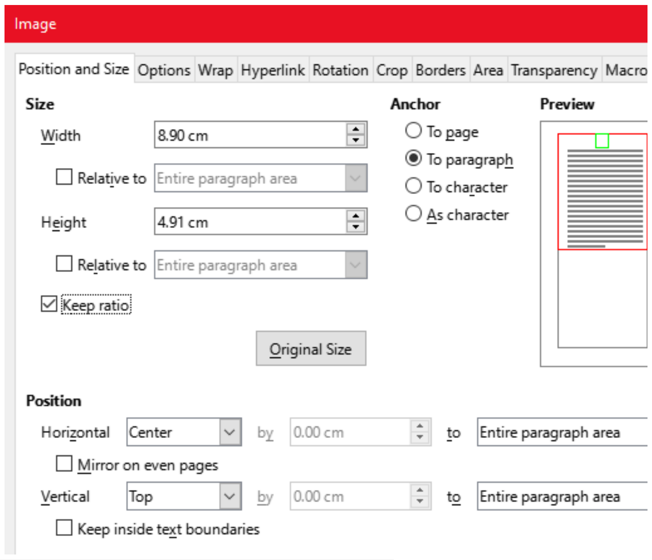


Image
Position and Size | Options | Wrap | Hyperlink | Rotation | Crop | Borders | Area | Transparency | Macro
Preview
Original Size

Size
Width: 8.90 cm
☐ Relative to: Entire paragraph area
Height: 4.91 cm
☐ Relative to: Entire paragraph area
☑ Keep ratio

Anchor
○ To page
● To paragraph
○ To character
○ As character

Position
Horizontal: Center by 0.00 cm to Entire paragraph area
☐ Mirror on even pages
Vertical: Top by 0.00 cm to Entire paragraph area
☐ Keep inside text boundaries

Figure 2: ImagePosSize.png

The writing style in VML for images is:

```
<:@image:./../img/dir/ImagePosSize.png::
id=__Img_ImagePosSize.png ::
title=Figure 1: ImagePosSize.png ::
style=ImageCenter :: size=8.5cm*7.26cm ::
px=512*437 :: DPI = 153.>
```

For translation VML to LibreOffice only the size information is relevant. But the height can be removed, the image is resized with its ratio automatically during translation.

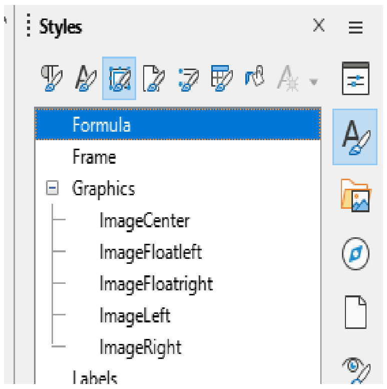
The title builds the content of the paragraph for the image caption, where the image is bounded to. The style of this paragraph is always either `ImgCaptionText` or also `mgCaptionTextPg` or `mgCaptionTextCol` depending from a

```
<:columnBreak.>
```

or a `<:pageBreak.>` before the image line separated with an empty line.

The given style `style=ImageCenter` determines the used style for the image itself.

Figure 3: ImageStyles.png



The right image has the style `ImageFloatLeft`, and that's why the text floats left of the image as seen here.

In profession, the saying *"less is more"* is important. Only a few scopes for design is really enough. That is for example a right side image flowing on left side with text (as usual in Wikipedia), an image positioned left side and flowing with text right side, or a central or left or right aligned non flowing image. The borders are not a point of discussion, borders should always the same, for example 2 mm or 0.08 inch. That suggest, using an indirect style also for images in LibreOffice and remove all direct styles.

The preferred styles in LibreOffice for images are:

- `Img`: A central image between paragraphs.
- `ImgRight`: A right side image between paragraphs.
- `ImgfloatLeft`: A right side image as part of a paragraph, floated left side with the paragraph's text.
- `ImgLeft`: A left side image between paragraphs.

- `ImgfloatRight`: A left side image as part of a paragraph, floated right side with the paragraph's text.
- `ImgChar`: An image inside of a line of the text of a paragraph, usual a small image.
- `ImgfloatChar`: An image inside of the text of a paragraph, the lines above are left and right of the image, the base line is broken by (contains) the image.

No more is necessary.

The following syntax is used for images in VML:

```
<:@image:PATH/TO/IMAGE ::
title=CAPTION ::
style=STYLE ::
size=Xcm*Ycm ::
px=PX::PY ::
DPI=DPI
```

All arguments are optional, except the `PATH/TO/IMAGE`: Line breaks are optional after the `::`.

- i`PATH/TO/IMAGE`: This should be recommended a relative path to the image starting from the odt document folder. You can use in MS-Windows a symbolic directory link created with `mklink /J NAME PATH` or also a symbolic linked directory in Linux/Unix to a little bit remote existing image directory tree to reach images with a simple link. Using an absolute path is strongly not recommended, because then, you cannot copy your files to another computer with a non exact equal directory tree structure. Also links inside the odt document are possible but not recommended.
- `title=TEXT`: A title or caption for each image should be recommended.
- `style=STYLENAME`: This should be one of the named indirect styles for the image positioning.
- `size=xSize*ySize`: The image size should be usual given in the measurements of the document, not in pixel. See 3.7.1 Some remarks to size of images. Write for example `size=9.87cm*3.14cm` or `size=123pt*87pt`. If this parameter is not given but : `DPI`=.. is given, then the size is calculated by this values.
- `px=xPixel*yPixel`: This value is used only if the image is not available as file while translation. Elsewhere the pixel size is read from the image file and write to VML as information. Note: Till now only png images are used.

3.7.1 Some remarks to size of images

For a printed document and also for pdf and inside LibreOffice the resolution of pixel depends on the output capability. It is not related to the pixel size of the given image file. The printer or render in pdf and inside LibreOffice adapts the pixel of the image to the pixel of the used output. For that also anti-aliasing algorithm are usual used. That's why the pixel size does not play a role for the size of the image. It may be interesting only for the resolution or quality of the image.

The size is determined by the size on the output device or related to the paper format. It is named in following text as "printed size". That is either a value in cm, inches, pt or pica as usual units for that. Only this size is used also internally for LibreOffice.

But, sometimes the pixel size should be used to determine the printed size, if the image is changed or if the document is written newly, maybe to show one pixel of the image exactly by 1, 2 or 4 pixel in the printed output, maybe to have a relation to the image pixel size, or maybe also to prevent some aliasing effects on bad rendering.

That’s why you can give the image size also in pixel with a related DPI (*"dot per inch"*) resolution. It the printed size as `size=...` is not given, then this printed size value is calculated by translation to LibreOffice.

For translation from `LibreOffice.odt` to `Plaintext.vml.adoc` the pixel size is gotten from the image file (if it exists in the given link), and the DPI value is calculated with the given print size. The DPI value may be an interesting information. With this information you can tune the size of the image in your `vml.adoc` file, for example tuning the DPI value, together with removing the information to force new calculation of the size.

For example you see the following line:

```
<:image:...
:: size=5.6cm*4.3cm :: px=1024::768 ::
DPI=464*453 .>
```

Then you see, you have a fine resolution, because the image is small with a high number of pixel, but you see also that the image is a little bit biased. The reason may be, the image was change in pixel size, but not in the document. If you change this line to

```
<:image:...
:: px=1024::768 :: DPI=450 .>
```

The you force new calculation of the size in cm, whereby the size will be a little bit greater, because of reduced DPI. But now the bias is removed, the original width and height relation is mapped. And last not least for a printing output with 150 DPI exact three image pixel are used to build the print pixel. On next generation from LibreOffice you will get the line

```
<:image:...
:: size=5.78cm*4.34cm :: px=1024::768 ::
DPI=450 .>
```

which is the real size now.

If the size is given in the

```
<:image:...
:: size=10cm*5cm .>
```

then this given printing size value is used, independent of the image pixel size. The additional pixel size and DPI value is ignored then.

3.8 Transcription of specific characters

There are some non or bad visible characters which needs transcription. This is:

Non breaking space

VMU	adoc	html	UTF-16	UTF-8
`\ `	`{nbsp}`	`&nbsp;`	`\u00a0`	`c2 a0`

LibreOffice: *“Insert – Formatting mark – Insert Non breaking space Sh-Ctrl-space”*.

Appearance inside a text only visible in LibreOffice itself: etc. pp.

The non breaking space is a normal space, in editors usual shown as a simple space, but a line break on this position is prevented. The transcription in VMU is simple, only a backslash before the space is written, to see it.

Using for example for *etc. pp*, which should not break the line between *etc.* and the following *pp*.

Zero width space (optional break)

VMU	adoc	html	UTF-16	UTF-8
`|`	`{zwsp}`		`\u200b`	`e2 80 8b`

LibreOffice: *“Insert – Formatting mark – No width Optional Break Ctrl-”*.

Appearance inside a text only visible in LibreOffice itself: A verylongword

A white space with an appearance of non space. But a line wrap can be inserted on this position if necessary.

word joiner

VMU	adoc	html	UTF-16	UTF-8
`\+`	`{wj}`		`\u2060`	`e2 81 a0`

LibreOffice: *“Insert – Formatting mark – Word Joiner"*.

Appearance inside a text only visible in LibreOffice itself: word joiner

A non visible character which prevents a line break on this position, though a breaking possible character follows, with the next following words (also after some spaces or tabs).

3.9 Using Character styles, semantic text span

4 Implementation

Table of Contents

4.1 WriteOdt

The main source for the writer can be found in org.vishia.odt.readOdt.WriteOdt (www). It contains the WriteOdt.main(...) (www) to start from command line. Parsing all command line arguments is done with the class org.vishia.util.Arguments (www) from the used base library vishiaBase, The main calls WriteOdt.smain(...) (www) and then WriteOdt.amain(...) (www) and then with already outside prepared arguments to support calling from a superior tool (for example a GUI). The execute(…) does the work.

The WriteOdt.execute(...) (www) reads firstly the content from a maybe given `-cfg:file` for some settings.

The `-odt:file.odt` should be first opened as zip file, to read out its given style.xml for checks. This should be done in Version-2, yet not.

The `-oxml:content.xml` is opened for writing. If this argument is not given, it is supplemented by a `content.xml` file either -in `dbgDir:dir` or in the `-i:input` directory. The org.vishia.xmlSimple.XmlSequWriter (www) is used for writing XML. This is a simple sequential writer which does not build a tree of XML data, instead writing as coming. Hence the order of elements for output is well proper.

First the name space and head information are written to the `content.xml`. Then WriteOdt.parseAdocWriteOdt(...) (www) is called which does the internal work. Then XML writing is finished and at last `content.xml` in the zip file `-odt:file.odt` is replaced.

WriteOdt.parseAdocWriteOdt(...) (www) works in the following kind:

It reads line per line the textual input file `-ifile.vml.adoc`. For any line parseAdocM(...) (www) is called. This checks the beginning of each trimmed line (left spaces are ignored). Note that lines of the elements before which continues the text are processed already, it means this operation sees the start of a new item of the text. This line starts are:

- `*` → parseList(...) (www)
- else, if the line does not start with `*` (www) , then a currently list is closed.
- `=` → writeHeaderLine(...) (www) : The line should contain with `=== Chaptertitle <:#Label>`
- `###` → Line is ignored, it’s a comment.

Note, this are the only one simple character on start of a line which triggers. All other are the <:xxx Designation.

- `<:p:` → parseWriteParagrStyleLabel(...) (www) The paragraph starts with a style definition in form <`:p:style.>`. The following text and following text lines builds the text of the paragraph. See also parseWriteText(...) (www)

- `<:Code:` → parseWriteCodeBlock(...) (www) and the text till `<.Code>` is parsed and translated.
- `<:table` → parseWriteTable(...) (www)
- `<:pageBreak.>` → It sets only the flag bPageBreakBefore (www) , recognized for the next style as modification (chapter title, paragraph).
- `<:columnBreak.>` → It sets adequate only the flag bColumnBreakBefore (www)
- `<:Section:` → parseWriteSection(...) (www)
- `<.Section>`: writeSectionEnd(...) (www)
- `<:TOC`: writeTableOfContents(...) (www)
- else → parseWriteParagr(...) (www) If non of this Designations met, the following text is a standard paragraph.

5 Hints to Asciidoc usage

5.1 Defining of own Css styles for Asciidoc

TODO it is a little bit confuse.

6 Hints to Libre Office usage

TODO

6.1 Exchange and maintain the styles of the document

It is planned that also the styles.xml inside the odg document should be convert to a plain text presentation and back again. If this is done, also the styles can be edited in the plain text and compare for versions.

Problem is that also too much styles are presented in the style list, sometimes irritating, sometimes useless styles, which can be set to “Hidden Styles” (cannot be removed).

The second problem is, some properties of styles are really derived, but they are not shown as derived. And this is an important thing.

It is possible to open the odg document as zip, copy the `styles.xml,` view it, edit it if you are familiar with the XML content, replace it. But make a save copy before and check whether all is ok after.

7 Internals

7.1 XML coding for internal references to bookmarks

TODO