Contents of this talk

Components all around

An interesting multi page document often contains some combination of:

- Text in chapters, sections and paragraphs.
- Graphics, figures and tables to elucidate and complement the text.
- For software engineers: code snippets to explain certain implementation details.

For all these components there are optimal applications. Editors to write text, Drawing or design tools to create graphics, pictures and design diagrams, spreadsheets to support with tabular material.

All in one file?

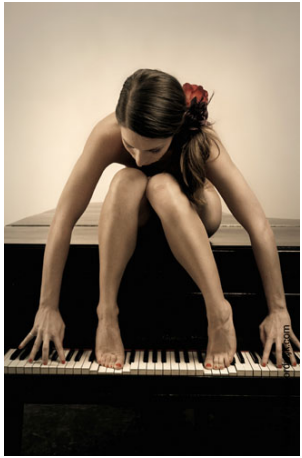
Putting all in one file is not always a good idea

Certainly not if you try to make a document with multiple authors. Try playing a piano “quatre main”. That is hard enough. Typing on a laptop with four persons is just monkey-business. So putting all in one document is not a good idea. Appreciate the possibility to keep the parts apart.

All in one file?

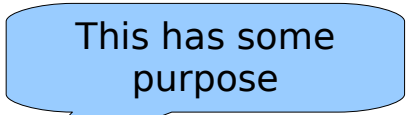
Putting all in one file is not always a good idea

Certainly not if you try to make a document with multiple authors. Try playing a piano "quatre main". That is hard enough. Typing on a laptop with four persons is just monkey-business. So putting all in one document is not a good idea. Appreciate the possibility to keep the parts apart.



Keeping the components separate

The example below is build out of a drawing file, that includes a table from a spreadsheet, included into this presentation but all kept as separate files. It thus allows three people to work on the components all at the same time.



	One	Two	Three
Top		1	
Middle			2
Bottom			3

Components allow you to share work

- In our courses, most of the project work is done in groups.
- Ideally you want all members to do an equal share of the work
  - Not just for fairness but...
  - because doing a piece of work makes you think and hence *learn*.
- Maintaining documents as components (that is: in separate files) makes this possible.

If you think that you all ready work like this, think again: Each time you have a correction of one of the parts, you will have to (partly) reassemble[1] your document. You do not want to do that very often. And there is no need to if you use the proper tools.

[1] you cannot reassemble automatically, because the editor will want to ask you if the links should be refreshed or something like that.

WYSIWYG

In the early days of word processing and graphical user interfaces, WYSIWYG was considered an innovation. And indeed, it helped the layman to create nicely looking documents. But WYSIWYG has a major flaw: it does not educate people and takes the focus to the wrong spots.

Very often, the tools are not use in a proper way. The possibility to use bold, font faces and sizes and what not to your harts content is really too tempting. The result is very often an amateurish looking document with all the wrong properties and very often missing essential features like page numbers. 

It is sad to see that nowadays many students hand in documents with this kind of 'quality'.  .

What you really need is

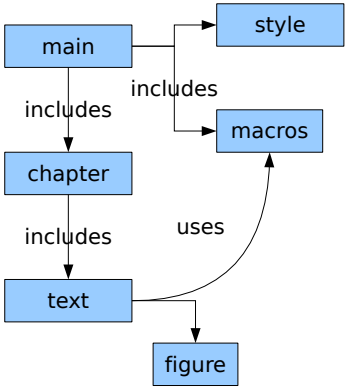
WYGIWYW

What You Get Is What You Want.

Working with multiple authors.

If you work in a team project, you want to share the document writing work also. It is the most boring part anyway .

In fact, this is similar to the way you create programs in modern OO-languages like **Java**and **Csharp**: Multiple small files, each with one purpose and responsibility. And it is ideal for group work, sharing and *reuse*.



Having components facilitates work sharing

If you work with a team of developers or document authors, you should maximize the amount of parallelism. This can be achieved by allowing all team members to work independently on separate parts:

- Each writer works on (creates or changes) a separate subset of the files.
  - It allows the author to concentrate on one issue at the time: content.
- At least the builder, (maybe chief editor) has (a recent) copy of all the components of the final product.
- It is advisable for all team members to have such a complete set of files. Use a version control system like subversion to your advantage.
- Building the final product is doing an update from the repository, then a build. (Make). From there you have a publishable product.

The components of a document

Lets delve into the content; A document has *structure elements* like

- Chapters
- Section
- Subsection
- Paragraphs
- Indexes and tables of contents
- Picture, maybe with caption.
- etc.

All these elements have an associated meaning to the reader, which is conveyed to the reader by formatting, layout and white space.

Separation of structure, content and formatting

The best known example of a potentially clean separation between content and structure on the one hand and formatting on the other is found in the combination of html+css. Proper use implies:

- The HTML code itself contains no formatting, only tags(=structure) +contents, combined with style classes and possibly use of the name or ids of the tags. Example: `<td class='date'>2007-08-31</td>`
- A separate style sheet defined the rendering of such an element. Example: `td.date {font-weight:bold; color:#080;}`
- Proper use of this enables separation of concerns and **reuse**. The same text could be used in a different context with a completely new style. It is even possible to have a personal rendering with some kind of pimped style for every user. Luckily this last is seldomly used.

*ML does not quite cut it

There are a few disadvantages to *ML which make it less usable for our multi writer purpose:

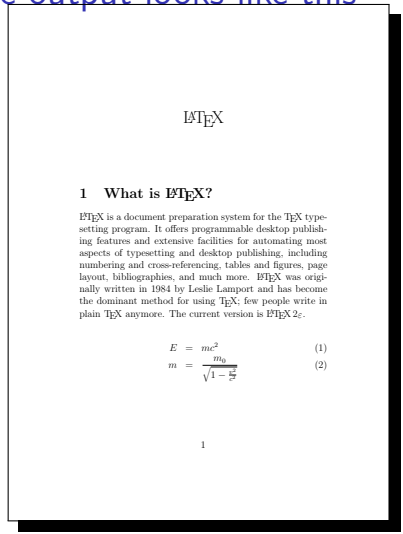
- Including from static files is awkward. There is no equivalent to the C++ *include* statement.
- The renderer (browser) does not[2] produce chapter and section numbers on its own or a table of contents. That would all be extra labor.
- The syntax is sometimes a bit verbose.
- However HTML can meet your needs if you add a small build script to it, or if you always produce it through a program like PHP or Java etc.

[2] HTML5 and CSS3 can do a lot

LaTeX as the project's document source format.

```
\documentclass[a5paper,12pt]{article}
\usepackage[a5paper]{geometry}
\title{\LaTeX}
\date{}
\begin{document}
  \maketitle
  \section{What is \LaTeX?}
  \LaTeX{} is a document preparation system for the \TeX{}
  typesetting program. It offers programmable desktop publishing
  features and extensive facilities for automating most aspects of
  typesetting and desktop publishing, including numbering and
  cross-referencing, tables and figures, page layout, bibliographies,
  and much more. \LaTeX{} was originally written in 1984 by Leslie
  Lamport and has become the dominant method for using \TeX; few
  people write in plain \TeX{} anymore. The current version is
  \LaTeXe.
  \newline
  % This is a comment, it is not shown in the final output.
  % The following shows a little of the typesetting power of LaTeX
  \begin{eqnarray}
    E &=& mc^2                                  \\
    m &=& \frac{m_0}{\sqrt{1-\frac{v^2}{c^2}}}
  \end{eqnarray}
\end{document}
```

The output looks like this



LaTeX

1 What is LaTeX?

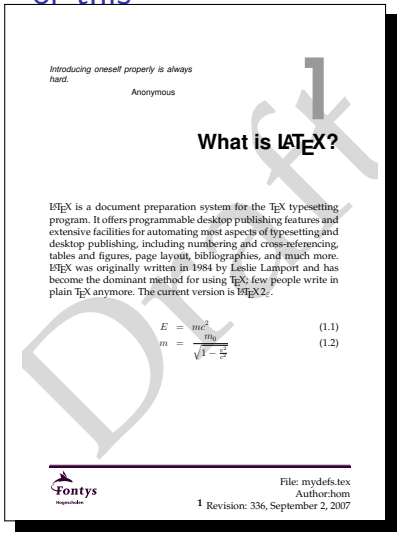
LaTeX is a document preparation system for the TeX typesetting program. It offers programmable desktop publishing features and extensive facilities for automating most aspects of typesetting and desktop publishing, including numbering and cross-referencing, tables and figures, page layout, bibliographies, and much more. LaTeX was originally written in 1984 by Leslie Lamport and has become the dominant method for using TeX; few people write in plain TeX anymore. The current version is LaTeX 2ε.

$$E = mc^2 \quad (1)$$

$$m = \frac{m_0}{\sqrt{1-\frac{v^2}{c^2}}} \quad (2)$$

1

or this



Introducing oneself properly is always hard.

Anonymous

1

What is LaTeX?

LaTeX is a document preparation system for the TeX typesetting program. It offers programmable desktop publishing features and extensive facilities for automating most aspects of typesetting and desktop publishing, including numbering and cross-referencing, tables and figures, page layout, bibliographies, and much more. LaTeX was originally written in 1984 by Leslie Lamport and has become the dominant method for using TeX; few people write in plain TeX anymore. The current version is LaTeX 2ε.

$$E = mc^2 \quad (1.1)$$

$$m = \frac{m_0}{\sqrt{1-\frac{v^2}{c^2}}} \quad (1.2)$$

Draft

Fontys Hogescholen

File: mydefs.tex
Author:hom
Revision: 336, September 2, 2007

Group work enablers

The following standard LaTeX features make group work a lot easier.

- Source formatting is similar to programming in some programming languages, including indentation.
- Comments are not rendered, but are a nice means to communicate with co-authors.
- Changes between versions are very easily checked (with standard diff tools and built in facilities of version control tools)
- Splitting and combining with **include** for chapters or **input** for everything else. This brings the power of reuse of macros and style sheets or other content (DRY principle). Try *replacing* one style with another in a word processing package and you know what I am talking about.

Project directory structure for small documents

In the software engineering team we have good experience with the following conventions for small[3] documents:

- All style files are kept in the root of the project directory or in a separate directory named **style**.
- Organise the document like in one file per (book) chapter.
- The name of the top level file is *main.tex*[4]. This file inputs all style definitions and the contents proper.
- The contents proper is defined in the file called *material.tex*

[3] single part reports

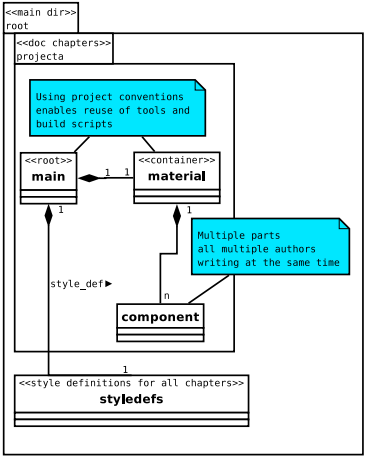
[4] Rename the main.pdf as last step

Project directory structure for big projects

- If the project documentation consists of (book) parts: One directory per part, allowing each part to be processed separately.
- All tables, figures, code snippets and other non *tex* source files are kept in a separate directory.
- In a bigger project, like SoFa, you will have multiple documents as deliverables, like project plan, requirements, analysis, design, test plan, user manual etc. These are all *part*s of a bigger document.
- In this case, choose a bit more elaborate structure, which in the end allows easier management.

Project document structure examples

UML for small project.



Dir struct for big project.

- 10project
  - ipso.tex
  - main.tex
  - material.tex
- 20requirements
- 30analysis
- 40design
- 50testplan
- 60implementation
- 70usermanual
- 80glossary
- All
- maintemplate.tex
- style.tex

Example main and material

main.tex:

```
\input{../style}
\begin{document}
  \input{material}
\end{document}
```

introduction.tex:

```
\section{Introduction}
The Super Project ....

The client ...
```

material.tex:

```
% include works also if the chapters
% are not yet ready.
% For personal builds use the
% \includeonly directive command
\include{introduction}
\include{componentbased}
\include{separationofconcerns}
```

What file formats can be used

- The main texts are written in a standard text editor.
- For English texts this is just fine, but if you need to use special characters with accent or umlauts, it is more convenient to use a so called input encoding.
- Quite usable are latin1 (which is equivalent to iso-8859-1) or UTF8. I have good experience with UTF8 and most Linux text editors.
- If possible, use an editor with a spelling checker. It is acceptable to use a word processor, as long as you store the text in a plain (.txt) text format.

File formats for graphics

For the non text material you can use the following graphics formats in decreasing preference:

PDF This format can be generated by many programs including open office and some (vector based) drawing programs. Pdf is a vector format inheriting a lot from postscript. If you have (single page) postscript file, you can convert it to pdf with `epstopdf`.

PNG or portable network graphics. This is a bitmap format. If the resolution if the file matches the resolution of the output device combined with the scaling, this produces good results. PNG can use loss less compression, it is free of patents and is sometimes claimed to be the successor of GIF.

File formats for graphics, comparison.

JPEG or joint photographic expert group format. As the name implies, this format is intended for photographs, that is, pictures with gradual transitions. JPEG uses lossy compression. Use it for photographs, almost nothing else, otherwise the compression and decompression produces visible ugly artifacts.

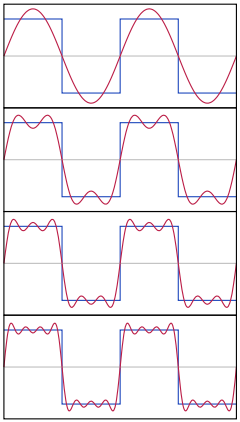
pdf: vectors always perfect

PNG at the wrong resolution

JPEG with its artefact's

Jpeg involves Fourier series...

Fourier series of a block wave



...and hence kaput for line graphics.

JPeg is invented to reduce file size of bandwidth (Digital video). The compression algoritm involves something similar to fourier series. `http://en.wikipedia.org/wiki/Fourier_series` and is lossy (Verlustbehaftet).

Converting the jpeg data back into a picture produces artefacts as can be explained from the left hand picture.

You do not need to be a LaTeX fundamentalist

Sometimes, getting things right with any program is hard, and in the beginning it is even harder with LaTeX. Although: most of what you type is text. And that really makes no difference. It only comes out better. Especially special constructs like intricate tables are hard to maintain with a simple text editor. Cheat where it is more efficient to do so. Examples:

- Create tables with a spreadsheet or a word processor and export it through PDF as single page graphics. Cut if the superfluous white space at the edges with *pdfcrop* and you will have nice tables with a lot less hassle.
- The same applies when you want to combine a table with graphics. Use something like **oodraw** and export to pdf. You have seen an example in some earlier sheet.
- Generate text for inclusion with a program like a python or Perl script. Reading from a database and outputting via LaTeX works wonderfully well.

Learning starts with copying

Teachers always give examples.

There are a lot of examples that you can learn from. The source code of this presentation for instance. You can find it and another latex example on `https://www.fontysvenlo.org/svnp/879417/latexcolloquium/trunk`, in one of my personal subversion repositories.

Thank you for your attention.