Manual for the `latex-maven-plugin` and for an according ant-task Version 2.0

Ernst Reissner (rei3ner@arcor.de)

2023-10-27

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

This document is created with `lualatex` or that like with output format pdf. The package `tex4ht` is not loaded.

LaTeX is a beautiful way to create printable documents, in our days preferably as portable document format (pdf)-files, with a particular strength in typesetting formulae like

$$\pi = \sqrt{12} \sum_{k=0}^{\infty} \frac{(-3)^{-k}}{2k+1}. \tag{1.1}$$

Here, portability of the format pdf is a vital feature. In the past, normally device independent; traditional output format of tex processors, today widely replaced by PDF (dvi) (device independent) described in [Rei17] has been used and still creation of external formats like HyperText Markup Language (html), open document text (odt) and current document format for MS Word (docx) are based on an intermediate dvi-file. It is much more lightweight than pdf specified in [ISOa], in [ISOb] and in [ISO12].

This piece of software implements both an ant-task and a maven-plugin generating documentation of various formats from LaTeX-files in a uniform way. Chapter 2 shows how to install both the maven-plugin and the ant-task and Chapter 3 describes the usage. Note that the maven-plugin is both easier to install and more versatile to be used.

From the LaTeX-files, the latex main files must be extracted, only these must be compiled. It is very usual to endow LaTeX-files with figures. On the other hand, there are many graphic formats which cannot be included directly in a LaTeX-file but must be preprocessed. If there is some format needed but not yet provided, please write an email to the author.

Graphic files must be preprocessed before processing latex main files, as described in Chapter 4. Then follows the proper processing of latex main files

including creation of index and glossaries as described in Chapter 5. Besides pdf, these formats include the web-formats html and extensible hypertext markup language (xhtml), open offices format odt, Microsoft's word formats like docx and finally plain text.

Uniformity of ant-task and a maven-plugin means in particular, that the settings which may be passed to the task and those allowed for the plugin are in a one-to-one relation. They are both described in Chapter 6. It is a design goal, that the auxiliary programs used by this software are fully configurable via parameters, that aspects not completely specified can be handled flexibly, there are parameters supporting information development and that for the parameters are default values which allow doing without explicit parametrization in most of the cases.Both, the ant-task and the maven-plugin rely on the same code base which form the package `org.m2latex.core`. The code specific for the ant-task is in `org.m2latex.antTask` and that specific for the maven-plugin is in `org.m2latex.mojo`.

The creation process supports an index, a glossary and a bibliography. In addition, code written in python and other languages can be included and executed during creation of the document. Again, further functionality can be added by demand.

The present manual is created by the maven-plugin or the ant-task described here. There should be no difference in the result. This manual is designed in a way that it covers the most important features but also to demand the most important features. That way, creating this manual is a top level test for the underlying software. The maven-plugin is somehow superior because it better supports the design process for the LaTeX sources.

If something goes wrong in the build process, or there is an indication of some deficiency in the result of the build process, processing must be aborted if going on does not make sense and there must be some error or warning logging as described in Chapter 7.

The author found some gaps, i.e. desirable features which are not yet implemented. To prioritize further work, all these gaps are collected in Chapter 8. Accordingly, the most important bugs are collected in Chapter 9. The user is encouraged to contribute with feature requests and bug reports and to vote for realization of features and on fixing bugs. Software quality is ensured mainly through tests which are described in Chapter 10.

Chapter 2

Installation

Both the ant-task and the maven-plugin just direct parameters from ant and from maven, respectively, to the programs that do the proper work. Thus, installation of the ant-task and of the maven-plugin requires that all needed programs are installed. These prerequisites are collected in Section 2.1.

2.1 Prerequisites

The ant-task is tested with

```
Apache Ant(TM) version 1.10.12 compiled on December 14 1969}
```

(of course the year is not correct, but this is the version string displayed by that release) and the maven-plugin with

```
Apache Maven 3.9.2
```

Both, ant and maven are written in java and require a java installation. The java version used for tests is `17.0.8`.

So, a java installation is the base for running either the ant-task or the maven-plugin. Also, this plugin is written in java. To use the maven-plugin, of course maven must be installed and to use the ant-task, ant must be installed.

The ant-task just passes parameters in the build file to the core and accordingly the maven-plugin passes parameters in the pom to the core of this software. The core just invokes various programs to do the actual work.

Besides plain building of documentation, this software also supports development of documents. LaTeX and related programs are based on text files mainly and so a good editor is required for development.

The author recommends and uses VS Code, e.g. 1.81.1 in conjunction with package LaTeX workshop 9.13.4 and LTeX 13.1.0.

An alternative is good old

```
GNU Emacs 24.3.1 (x86\_64-suse-linux-gnu, GTK+ Version 3.16.7)
```

together with several packages to support various file formats. To list the available packages type `M-x list-packages`. For comfortable development with LaTeX, the `auctex` package, version `11.88` is recommended. The version is displayed from within Emacs by typing `C-h v AUCTeX-version RET`. For an overview on `auctex` see [TAK+14].

FIXME: gnuplot-mode expects file extension gp. Should be made configurable.

To edit metapost, the mode built-in mode `Metamode` is used.

Built-in mode `Docview` to view pdf, ps and dvi.

latexmk

Builtin modes bib-mode and bibtex

Built in reftex-modes

Useful: ac-math, auto-complete-auctex

Depending on what kinds of graphic formats are used, the following programs are required:

- To convert the native file format for xfig (fig)-files into pdf-files, by default `fig2dev` is used. It makes sense to have `xfig` installed to create and edit fig-files, but this is not mandatory.

- To convert gnuplot files into PDF-files, there is no alternative, to have installed `gnuplot`. It serves as an interpreter and also as a converter. Strictly speaking, only the latter functionality is required here.

- To convert metapost: input format for the graphic program `mpost` (mp)-files into encapsulated postscript (eps)-files, the interpreter `mpost` or equivalent is required. This comes with a standard tex-installation. With the standard configuration, the resulting EPS-file can be viewed with `ghostscript` and for developing it is recommended to have `ghostscript` installed.

- To include Scalable Vector Graphics (svg)-files into LaTeX, `inkscape` must be installed. It also serves to create and to edit SVG-files.

Currently, for including PDF-files in both cases, the driver `dvipdfmx` must be installed. Strictly speaking, this is required only for HTML-creation and related. Note that if no pictures created by `fig2dev`, `gnuplot`, `mpost` or by `inkscape` are used, of course, neither `fig2dev` nor `gnuplot`, `mpost`, `inkscape` nor `dvipdfmx` is needed. To include graphics, the graphics bundle described in [Car16] is required, except for SVG-files which requires the svg-package described in [Ilt12].

As the set of required software depends on the graphic formats which shall be imported, it depends also on the set of output-formats to be supported:

- To create PDF-files from LaTeX-files we use `lualatex` or some other kind of LaTeX creating PDF-files like `xelatex` or `pdflatex`.

  LaTeX uses several auxiliary programs. Above all `bibtex`, to create the bibliography and `makeindex` and `splitindex` for the index and `makeglossaries` for the glossary. The latter two also require the latex packages `makeidx`, optionally `showidx`, both described in [BLC+14], the package `splitidx` documented in [Koh16] and `glossaries` specified in [Tal16]. Note that `makeglossaries` either invokes `makeindex` or `xindy`, depending on the parametrization of `glossaries`. Both, `makeglossaries` and `xindy` are written in Perl, which shall also be installed if a glossary is required.

  To include program code in Python, octave and other language, `pythontex` is needed; to eliminate that code creating an equivalent tex-file, one has to combine it with `depythontex`. Both are written in Python3 which shall be installed also as a dependency. To use them, one also needs to install the package `pythontex`.

  The package `rerunfilecheck` is in any standard LaTeX-installation. It is almost mandatory because this software presupposes that package is present to ensure that the table of contents, list of figures, list of tables, the index and the glossary are up-to-date.

  It is standard to endow a PDF-file with hyperlinks. To support this, the package `hyperref` is required.

  ****

- To create html-files, or to be more precise any kind of Standard generalized markup language (sgml) and extensible markup language (xml), from LaTeX-files, `htlatex` or alternatively `htxelatex` is used. Currently, the author is not aware of any alternative to the two. This includes also creating OpenOffice documents like ODT-files. Thus, OpenOffice documents are created in two steps, the first is to create PDF-files with the according tools, the second one is done by `htlatex` or that like.

- To create RTF-files, currently `latex2rtf` is used. Note that this does not require `pdflatex`. As a drawback, not all LaTeX-packages are supported.

- MS Word documents are created from OpenOffice documents via the command `odt2doc` and thus require three steps and so the according tool chain.

- Finally, there is a way, to create plain text files from the PDF-files via `pdftotext`. The way from LaTeX to text via PDF makes sense because that text is well formatted math mode symbols like π and because table of

contents, index, glossary and that like are included. So, for that task, besides `pdftotext` the whole tool chain to create PDF-files is required.

- An application which does not create a target, i.e. a file in the target directory is `chktex` which just checks the latex main files and associated files.

So to run this software, the aforementioned programs or at least the subset used, must be installed. To obtain reproducible results, the versions must fit. This version is checked with the executables with versions given by an according properties file version.properties.

There are also several LaTeX-packages needed or at least recommended. The recommended ones are

- `geometry` described in [Ume10] to control page layout.
- `microtype` described in [Sch16] improve readability and make the document look nicer. It also helps to avoid bad boxes.
- `hyperref` described in [RO22] to insert hypertext marks, which I do not want to miss in larger documents.
- `srcltx` described in [SU06] which allows jumping from the DVI file to the TEX source and back.
- `showframe` if `geometry` is not used with option `showframe`. There seems to be no package documentation for package `showframe`.
- `booktabs` described in [Fea16]
- `fix-cm` described in [SMCR15] and `anyfontsize` described in [Sza07] to allow arbitrary font sizes, eliminating certain warnings.

Almost required are

- `rerunfilecheck` described in [Obe16b] which writes additional rerun warnings to the log file if some auxiliary files have changed. This software relies on these warnings to control rerun latex and other applications.
- `ifthen` described in [Car14] which provides the `ifthenelse`-command which is needed to create both PDF and HTML and also to create RTF.
- `iftex` described in [Tea22] which has two functions:
  - It provides the `\ifpdf`-command to detect pdf-mode. This is required to distinguish creation of PDF and text from HTML, ODT, DOC and others, based on DVI/XDV.

  - Also, it is able to detect a specific latex engine via commands like `\ifluatex` or `\ifpdftex` but also `\iftutex` being true for `lualatex` and `xelatex`, but not for `pdflatex`. This is used if a document shall work for more than one engine like this manual and is in particular used to create reproducible PDF files which is engine specific. Finally, there is a way to force an exception if the wrong engine is used, e.g. by specifying `\RequireLuaTeX`.

- The graphics packages described in [Car16], in particular `graphicx`, `xcolor` and `transparent`, the latter two described in [Ker16] and in [Obe16c], respectively. Sometimes also `bmpsize` described in [Obe16a] if pixel graphics is used.
- `import` described in [Ars09] e.g. to import nested graphic files from arbitrary directories.
- `inputenc` described in [JM15] to select an input encoding `fontenc` to select a font encoding. Font selection is described in [Tea00] in general, with Section 5 on font encoding and Section 5.1 on the `fontenc` package. This package is almost indispensable if you do not write English, e.g. to access German umlauts. Note that [MFL16] describes font encoding in more detail.
- `makeidx` and `showidx` described in [BLC+14] or something comparable for creating indices.
- `glossaries` described in [Tal16] with tutorial [Tal21] or something comparable for creating glossaries.
- `tocbibind` described in [WP10] to include bibliography and index (what about glossaries?) into the table of contents.
- `nag` described in [Sch11] which performs certain checks unveiling deficiencies not filtered by the compiler nor by another check tool.
- `babel` described in [BB16] for language support. This is not used by this manual, because it is in English.

Useful packages with which this software is tested:

- The ams-packages **** `amsmath`
- `longtable` described in [Car98] for long tables, i.e. tables exceeding a page.
- `listings` described in [HMH15] for listings.
- `fancyvrb` described in [Zan10] provides useful environments to mark verbatim text.

```
<project ...>
  ...
  <repositories>
    <repository>
      <id>publicRepoAtSimuline</id>
      <name>repo at simuline</name>
      <url>https://www.simuline.eu/RepositoryMaven</url>
    </repository>
  </repositories>
  ...
</project>
```

Listing 2.1: The source repository for this plugin

2.2 Setting up pom.xml for the maven plugin

If this software is used as a maven plugin, it need not explicitly be installed, maven itself does this by need based on the entries of the pom.

We can distinguish between entries in the pom which are necessary for any kind of usage of this maven plugin described in Section 2.2.1, configuration settings to obtain behavior deviating from the default as sketched in Section 2.2.2 and finally executions to integrate this plugin in the lifecycle as described in Section 2.2.3.

2.2.1 Basic setup

Unfortunately, this plugin did not yet make it into maven central. Thus, one has to add the providers' repository to the pom as shown in Listing 2.1 to enable maven to find the software.

Then just adding the coordinates in the build-plugins section of the pom as shown in Listing 2.2, and it can be used from command line, e.g. to create PDF files as `mvn latex:pdf` or for cleanup `mvn latex:clr` with default configuration. Alternatively, the plugin can be inserted also in the reporting-plugins section of the pom.

This plugin is indexed in `https://mvnrepository.com/artifact/eu.simuline.m2latex/latex-maven-plugin`.

2.2.2 Deviating from default settings

This plugin is highly configurable by means of a lot of settings. Listing 2.3 shows some of them, but all are in their default settings, so no need to specify them explicitly:

```
<project ...>
  ...
  <build>
    ...
    <plugins>
      ...
      <!-- create html and pdf and other formats from latex -->
      <plugin>
        <groupId>eu.simuline.m2latex</groupId>
        <artifactId>latex-maven-plugin</artifactId>
        <version>2.0</version>
      </plugin>
      ...
    </plugins>
    ...
  </build>
  ...
</project>
```

Listing 2.2: The coordinates of this plugin

targets defines the output formats and the checks to be run (`chk` signifies running `chktex`) for goal `cfg`.

cleanUp whether all generated files shall be removed leaving the LaTeX source folder untouched.

latex2pdfCommand is one of the names of the tools to be invoked. There are more settings for treating tools.

It is the experience of the author, that in many situations no explicit setting is necessary at all. The only setting needed to be configured regularly is `targets` which determines the output formats and whether sources are checked for goal `cfg`. It is recommended to use checking via target `chk` in any case. Some settings are only relevant only for document development as described in Section 3.5, one of these is `cleanUp`: setting this to `false` keeps intermediate files, in particular log files available which helps to find errors and in locating the sources of warnings. There are further situations where the user is grateful of being able to configure this software, or even where it is not usable with default settings. Chapter 6 describes the complete set of settings. The same pieces of information is given in the pom.xml used to test this plugin. Although each setting takes its default value, it is given explicitly and is endowed with a comment describing it in detail as in Chapter 6. Since this pom is quite long, it is not part of this manual but is given by reference on the project site.

```
<!-- create html and pdf and other formats from latex -->
<plugin>
  <groupId>eu.simuline.m2latex</groupId>
  <artifactId>latex-maven-plugin</artifactId>
  <version>2.0</version>
  <configuration>
    <settings>
      <!--targets>chk, pdf, html</targets-->
      <!--latex2pdfCommand>lualatex</latex2pdfCommand-->
      <!--cleanUp>true</cleanUp-->
      ...
    </settings>
  </configuration>
</plugin>
```

Listing 2.3: The coordinates of this plugin and some settings

2.2.3 Executions

To make the plugin available within a build, one has to add executions, e.g. as shown in Listing 2.4. For all goals specified there, a default phase is defined, as given as a comment but as this is hard-coded, one has to specify in the executions only when deviating from the default.

Typically, this plugin, to be more precise its goal `cfg`, which allows configuring checks and the output formats in setting `targets`, is used in the `site` lifecycle phase to process latex sources. It is perfectly ok to stick to a single format like `pdf` and configure `target` accordingly.

Alternatively, one may define an execution with the required goals like `pdf`, but then the phase must be specified explicitly, because there is no default phase. Of course, then no additional check is performed.

The goal `inj` injects files into the working directory `texSrcDirectory` as described in detail in Section 3.4. For some files it makes sense to do this independent of the build process e.g. by invoking `mvn latex:inj`, but in general it is preferable to perform the injection each build process anew because that way the injected file can be adapted to the current configuration of this plugin.

Normally, all created files in the source directory are removed after the output has been copied to the target, but during document development, described in Section 3.5, cleanup may be deactivated by setting `cleanUp` and so the source directory may be polluted. This may happen if other tools are used in conjunction with this plugin.

Nevertheless, cleanup is recommended to make the individual runs of this plugin independent. Thus, for document development, there is a dedicated goal `clr` to clean up the source directory in phase `clean`. Note that also the configuration files

created by goal `inj` are cleared.

Finally, it is recommended to add a check of the converter versions right in the phase `validate` via goal `vrs`. Listing 2.4 specifies `versionsWarnOnly=true`, which restricts goal `vrs` to just display a warning if something is wrong which seems appropriate in the context of validation.

For the default `versionsWarnOnly=false`, the goal `vrs` yields a full list of registered converters, signifying which one may cause trouble because its version is out of range as displayed in Listing 3.5. In the course of a build run, this seems too much information, but in fact, it is just a matter of taste.

For details on executions of goals `inj`, `clr` and `vrs` see Section 3.5.

The executions considered so far are appropriate for mavens default lifecycle. Typically, this maven plugin is used in the site lifecycle, which does not contain the phase `validate`, but accordingly `pre-site`. As a consequence, goals `inj` and `vrs` are not invoked. To get around, one could specify the phase `pre-site` in the execution explicitly. The author uses the `maven-jxr-plugin` as illustrated in Listing 2.5, which, as a side effect, forks the lifecycle and includes phase `validate` of the default lifecycle and in particular goals `inj` and `vrs`.

Note the section `configuration` in Listing 2.4 which is empty and can be skipped in a default configuration creating output in formats PDF and HTML and performing checks on the LaTeX-sources. However, pom.xml gives an example pom using this latex plugin with full configuration with default values and executions. In addition, Chapter 6 describes each setting individually.

2.3 Setting build.xml for the ant task

As you can see, the `taskdef`'s refer to java classes. Unlike maven which loads jars with the classes inside automatically from

```
https://www.simuline.eu/RepositoryMaven%
```

the jar for the tasks, `latex-maven-plugin-2.0-antTask.jar`, must be downloaded manually from

```
https://www.simuline.eu/RepositoryMaven/eu/simuline/m2latex/latex-maven-plugin/2.0%
```

Moreover, ant expects to find the jar files in an according folder. In my installation it is `/usr/share/ant/lib/`; as can be seen in the ant documentation, in general it is in folder `lib` in ant's installation directory.

However, build.xml gives an example build file using this latex ant task with full configuration with default values and executions. From that, one has to copy the following into the `build.xml` file in the current project:

```
<plugin>
  <groupId>eu.simuline.m2latex</groupId>
  <artifactId>latex-maven-plugin</artifactId>
  <version>2.0</version>
  <configuration>
    <settings>
      <!--targets>chk, pdf, html</targets-->
      <!--cleanUp>false</cleanUp-->
      ...
    </settings>
  </configuration>
  <executions>
    <execution>
      <id>process-latex-sources</id>
      <!-- chk, dvi, pdf, html, odt, docx, rtf, txt -->
      <goals><goal>cfg</goal></goals>
      <!-- phase>site</phase-->
    </execution>
    <execution>
      <id>clear-latex-sources</id>
      <goals><goal>clr</goal></goals>
      <!-- phase>clean</phase-->
    </execution>
    <execution>
      <?m2e execute onConfiguration?>
      <id>inject-files</id>
      <goals><goal>inj</goal></goals>
      <!-- phase>validate</phase-->
    </execution>
    <execution>
      <id>validate-converters</id>
      <goals><goal>vrs</goal></goals>
      <!-- phase>validate</phase-->
      <configuration>
        <versionsWarnOnly>true</versionsWarnOnly>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Listing 2.4: The executions of this plugin

```
<project>
  ...
  <reporting>
    <plugins>
      ...
      <!-- as a side effect,
           triggers 'generate-sources' forked phase execution -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jxr-plugin</artifactId>
        <version>3.3.0</version>
      </plugin>
      ...
    </plugins>
  </reporting>
</project>
```

Listing 2.5: Forked execution with jxr plugin

- The properties `antJarDir` and `createdJar`,
- The path element with the id `latex.classpath`
- The taskdefs `latexCfg` and `latex:Clr`
- The targets `latex:cfg` and `latex:clr`

As for the maven plugin, for the ant task, add configuration, where a deviation from the default requires to do so. The configuration is the same and is described in detail in Chapter 6.

2.4 Installation from source

The first step to install from source, is to clone from the repository by

```
git clone https://github.com/Reissner/maven-latex-plugin
```

of course assuming that `git` has been installed. Then change into the root repository where `pom.xml` for maven and also `built.xml` for ant are located.

To install the maven-plugin, ensure that maven is installed. One is tempted just to type

```
mvn clean install
```

but this does not work since the plugin needs itself to be installed to perform even `clean`. To solve that problem just comment out all its executions in the local `pom.xml` by enclosing them in `<!--…-->`. In fact this is a minor bug, since, to be strict, only the executions for verification and clearing must be deactivated. For processing, it would be sufficient to add `<phase>site</phase>` to execution `process-latex-sources`.

Since the author develops with maven, including the development of the ant task, the maven built, creates the file `latex-maven-plugin-2.0-antTask.jar` defining the ant task. To this end, also `mvn clean package` is sufficient. After that, installation proceeds like described in Section 2.3 copying that jar file ant's lib-folder where ant can find it.

With root access and after having checked the proper paths, the build file `build.xml` can be used to perform copy task by `ant install`, to insert an according link by `ant link` to remove it again with `ant uninstall`. The build file `build.xml` works only if `latex-maven-plugin-2.0-antTask.jar` is placed where ant can find it or if the parts are deactivated below the line

```
<!-- deactivate the following,
unless the ant task is installed already -->
```

I feel building with maven and linking the jar created is a very good way to develop the ant task, because after changes the new ant task is available immediately.

For typical changes in the sources, it is possible to recompile and package the ant task by `ant jar` also cleanup is possible with `ant clean`. Finally, the ant task can be tested with `ant latex:cfg` and `ant latex:clr`.

In the long run, it should be possible to build the ant task from sources with ant alone.

Chapter 3

Usage of Plugin and Task

This software offers both, a maven plugin and an according ant task, but the emphasis is on the maven plugin. Thus, the sections of this chapter are either general or apply to the maven plugin; only Section 3.7 specifically refers to the ant task. Usage presupposes installation as described in Chapter 2 including settings in `pom.xml` as described in Section 2.2 for the maven plugin and the settings in `build.xml` as described in Section 2.3 for the ant task.

This plugin may be used both if the LaTeX-sources are ready to create "final" output from them and also to support development of the LaTeX sources. Accordingly, this chapter has Section 3.1 devoted to the form of the sources, including directory structure, LaTeX-files and others, mainly graphic files included and a Section 3.2 on exporting into various formats.

There is a very special usage, called development of documents, which means while the document is under construction. The features and goals tied to this phase are collected in Section 3.5.

In contrast, Section 3.6 is on usage of the maven plugin within the lifecycles. This can be used during development of documents but is more appropriate for small changes or when development finished at a stage.

3.1 The source files

The LaTeX-files and also files included via `\input` are searched in the *TEX source directory* and subdirectories recursively. By default, this is `./src/site/tex`, where "`.`" is the *base directory* of this maven-project. The LaTeX-files to be compiled top level, typically not inputted anywhere via `\input`, are called *LaTeX main files*. As an example, in the *TEX source directory* of this software, `manualLMP.tex` is a LaTeX main file, whereas the file `header.tex` is not, although also a LaTeX-file. LaTeX main files are detected automatically.

The included files may be again be LaTeX-files, but also bibliography files, listings included by package `listings`, verbatim text included with `verbatim` and may be even code files to be executed via the package `pythontex`. The great bulk of input files however, are graphic files in various formats. As regards the way the according files are included in LaTeX-files, there are the following kinds of graphic formats, all included in the TEX source directory.

1. The first can be included into LaTeX-files directly via `\input`. These formats are essentially LaTeX and are defined in an according package. Examples are `eepic` described in [Kwo88] and above all `tikz` described in [Tan15].

2. The second one via the command `\includegraphics` defined by the package `graphicx` which is described in [Car16]. Chapter 2 therein mentions the supported drivers, among these are also `dvipdfm` and `dvipdfmx`. It is not the package but the driver which decides on the support of graphic formats. The dvipdfm user manual, [Wic99] lists the allowed formats MetaPost (i.e. metapost's postscript like output including text (mps)), postscript, pdf, Graphics format developed by the Joint Photographic Experts Group (jpg) and Portable Network Graphics (png).

3. The third one must be transformed into a graphics format of one of the former two kinds using an external tool for transformation. Here, of course, only a limited support is possible, because there is a broad variety of formats. We have chosen

   - the fig-format described in [Rei16] because of its simplicity,
   - the gnuplot format, described in [WK20], because it allows computation of function plots,
   - scalable vector graphics svg-format specified in [Da11][1] as it is important for construction and the counterpart of pixel oriented formats,
   - likewise, metapost (mp-format), described in [Hob14] because it is native to LaTeX and quite versatile

4. The fourth kind of graphics formats has to be transformed into one of the kinds one or two but unlike in type three, this is not done explicitly by an external tool but by a latex-package during the LaTeX-run. Note that, although not required to be explicitly transformed, those graphics files induce additional files by running LaTeX. Essentially, each of the abovementioned type of format can be included that way but currently, this is done for the svg-format only included by the package `svg` (see [Ilt12]). The author personally

[1] As the specification is hard to digest, we refer to the tutorial [DHH02].

refrains from using packages like that because of the lack of flexibility and further drawbacks.

5. Finally, there is a way to include graphics which is not really a graphic format: In the course of running code, e.g. by package `pythontex` in Python, as described in Section 5.5, it is also possible to create computed graphics. It may be advisable to separate code into special files to be included via `\input`, but it is not strictly required. In the long run it seems a good idea, to extend `pythontex` to read in code files, e.g. in python directly.

Note that unlike former versions of this software, the current version does not create a working directory by cloning the TEX source directory. Instead, it operates directly on the TEX source directory also creating intermediate files. The advantage of processing that way is, that this allows cooperation between this software and other tool chains which are better suited for developing latex files. Details are described in Section 3.5.

The downside is that a file residing in the TEX source directory risks being overwritten by this software, if it does not stick to the rules. The rules are simple: For each graphic file, being transformed, i.e. of types 3 or 4 above, additional files are created with the same name up to the suffix. Thus, for these graphic files no file with the same name up to the ending is allowed. The same is true for the LaTeX main files.

Besides the LaTeX-files and the graphics files there is a third kind of file supported: Bibliographies in bib-files. This software treats them automatically.

3.2 Exporting in various formats and checking sources

After having added the configuration of the plugin to the `pom.xml`, minimally the one given in Listing 2.2, it can be used directly invoking maven through `mvn latex:cfg`. Here `latex` is the (short) name of the plugin and `cfg` is the goal. It can also be interpreted as `mvn <source>:<target>`: The source files are in `latex`-format and the target are read from the *configuration* in the pom (*configuration* is what `cfg` stands for) which is illustrated in Listing 2.3.

By default, the targets are `cfg`, `pdf` and `html`. The following Listing 3.1 shows a configuration with the full range of output formats including in addition the OpenOffice document format `odt`, the MS word-formats `doc(x)` and `rtf` and also plain text format `txt` in utf8 encoding.

Note that the target `docx` converts by default into docx but may also be configured to produce the old-fashioned outdated document format for MS Word (doc) format.

```
<!-- create html and pdf and other formats from latex -->
<plugin>
  <groupId>eu.simuline.m2latex/groupId>
  <artifactId>latex-maven-plugin</artifactId>
  <version>2.0</version>

  <configuration>
    <settings>
      <targets>chk,pdf,dvi,html,odt,docx,rtf,txt</targets>
    </settings>
  </configuration>
</plugin>
```

Listing 3.1: Configuration with full range output formats

Be aware that the target `dvi` creates output in DVI format only for latex processors `lualatex` and `pdflatex`, whereas `xelatex` creates the XDV (extended DVI) format for target `dvi`.

Somehow special is the target `chk` which is mere checking without resulting output file. It just displays a warning if a rule is violated.

The resulting files in the given output formats are copied to the site directory, which is `./target/site` in a default maven project.

Sometimes it is more convenient to specify the output formats not via the pom but directly as a goal on the command line. In particular, one may write `mvn latex:pdf` to create documentation in PDF-format only. Likewise, command `mvn latex:dvi` to get good old dvi/xdv files or even `mvn latex:txt` for plain text, just as examples. Accordingly, `mvn latex:chk` performs a pure check. This occurs preferably in the context of documentation development. In particular, checking is treated separately in Section 3.5.3.

Note that the `-X` switch activates debugging which results in a more verbose output. Example: `mvn -X latex:cfg`.

In a standard maven project, the above minimal configuration should be sufficient. Only if the folder structure deviates from the standard or if the LaTeX sources require special configuration, parameters have to be given explicitly, because they deviate from the default values. Chapter 6 summarizes all available parameters, giving the default value and a description.

For sake of uniformity, the name of the ant-task is `latex:cfg`, and it can be invoked via `ant latex:cfg`. Unlike the maven-plugin, the ant-task does not allow to specify a target on the command line. The `-d` switch activates debugging which results in a more verbose output. Example: `ant -d latex:cfg`.

Whereas by default the target directory and in particular the target site directory

with all output of this plugin is deleted in maven's `clean` life-cycle, the tools invoked by this software also create intermediate files in the source directory. By default, i.e. for setting `<cleanUp>true</cleanUp>`, all files created in the source directory in the last run are cleaned. Nevertheless, for document development intermediate files are vital and so cleanup is frequently set to false. In this case, cleanup must be done in a separate goal, described in Section 3.5.5.

By default, the goal `clr` is also executed in maven's `clear` life-cycle.

There is an according ant-task `latex:cfg` which can be invoked via `ant -d latex:cfg`. FIXME: ant `latex:clr` has duplicate parameters. This can be fixed only by properties. Another problem is, to provide a complete subset of parameters which apply to `latex:cfg` and to `latex:cfg`, respectively.

3.3 Checking versions of converters

3.4 Injection of files

The goal `inj` is to inject files into the working directory `texSrcDirectory`.

Name	File	explanation
latexmkrc	`.latexmkrc`	config file for `latexmk`
chktexrc	`.chktexrc`	config file for `chktex`
header	`header.tex`	header file for latex sources
vscodeExt	`instVScode4tex.sh`	shell script to install VS Code extensions

Table 3.1: Overview over all injections

Table 3.1 shows the possible injections and the ones really to be performed are given in the configuration `injections`. The configuration is described in Table 6.2 on page 101. It is a comma separated list and the default is `latexmkrc,chktexrc`.

The file `.latexmkrc` tied to the injection `latexmkrc` is the configuration file for the build tool `latexmk` and likewise `.chktexrc` tied to the injection `chktexrc` is the configuration file for the style check tool `chktex`. Both tools are mainly used in document development as described in Sections 3.5.2 and 3.5.3, but both tools may be used in a regular build also. So their respective configuration files must be injected in the build process before the LaTeX build tools are invoked. Thus, goal `inj` has default phase `validate`. Note that `.latexmkrc` is adapted to the configuration of the current build, so that build with this software has the same result as direct invocation of `latexmk`. In the current version of this software, the injected `.latexmkrc` is not completely adapted to the configuration, and it is only intended to create PDF files. In the future this shall change.

The injection `header` is tied to the file `header.tex` which is intended to be included in each LaTeX main file. Essentially it includes packages always needed. It is inspired by the packages `pandoc` includes by default according to `https://pandoc.org/MANUAL.html#creating-a-pdf`. As the configuration files described above, it is intended to be injected in phase `validate`.

In the long run it could be adapted to the configuration as `.latexmkrc`, but currently it detects the use case as the latex converter or the target format and loads the according packages. Unlike `.latexmkrc`, it is not restricted to creating PDF. It is checked in conjunction with document classes `book` and `article`. **** TBD: mention also beamer class. ****

The injections described so far are intended to be performed in phase `validate` as illustrated in Listing 2.4. This is preferable, because the configuration files are adapted to the settings in each run.

The last injection, `vscodeExt` injecting the file `instVScode4tex.sh` is used in a completely different way. If the editor VS Code is already installed, the script `instVScode4tex.sh` also given by Listing 3.2, installs and updates all extensions the author used to write LaTeX-code.

Pasting .latexmkrc, which is just Perl code, into VS Code one can see the highlighting, and a preview, of course provided the extensions given by Listing 3.2 are installed; The Perl script `.latexmkrc` is in fact the configuration file for the development tool `latexmk`. It is adapted to the settings of this plugin. Thus, by default it is created with each run if not present and is erased when cleaning.

The file `.chktexrc` is to configure `chktex`. Its current state is given in Listing 3.3 and online. The user is kindly asked to help to improve it.

It is observed that the packages loaded by various LaTeX files have a huge overlap and that at the same time, although rare, exotic packages are loaded which may be replaced by standard ones. This hurts single source principle and at the same times makes it almost impossible for a build tool as this one, to make guarantees that it works still with the unexpected packages. This is, e.g. because a package may write warnings in an unexpected way into some log file.

Thus, a quite generic header file header.tex is also provided. It is written in a way, that it works for various generators, not only for `lualatex`. With some restrictions it supports generation not only of PDF files but also of HTML. There would be a lot more to say, but have a look at it yourself.

All these files are not only provided in the project site, but also by the injection goal of this plugin.

Thus, they are available in a project using this latex plugin using the dependency plugin. The first `artifactItem` extracts `instVScode4tex.sh`. Since `instVScode4tex.sh` is not only to install extensions in VS Code but also to update them, it is convenient to have the script at hand. Project `https://github.com/R`

```
#!/bin/sh -
# injection file written by latex plugin 2.0-SNAPSHOT

# USAGE:
# without argument defaults to `code`
# with single argument giving the name of the clone of code.
# Currently we use `CODE' for the flatpack version.
# This is used by eu.simuline:qMngmnt:0.0.6

# Explanation
# code --extensions-dir <dir>
#      Set the root path for extensions.
# code --list-extensions
#      List the installed extensions.
# code --show-versions
#      Show versions of installed extensions, when using --list-extension.
# code --install-extension (<extension-id> | <extension-vsix-path>)
#      Installs an extension.
# code --uninstall-extension (<extension-id> | <extension-vsix-path>)
#      Uninstalls an extension.
# code --enable-proposed-api (<extension-id>)
#      Enables proposed API features for extensions.
#      Can receive one or more extension IDs to enable individually.

case $# in
  0)
  code="code"
  ;;
  1)
  code=$1
  ;;
  *)
  echo "expected 0 or 1 arguments found $#."
  exit
esac
echo $code

echo "latex and friends"
# latex and friends
$code --force --install-extension james-yu.latex-workshop
$code --force --install-extension mathematic.vscode-latex
# lua
# [lua]: Couldn't find message for key config.runtime. ...
#code --force --install-extension sumneko.lua

# bib
$code --force --install-extension phr0s.bib
$code --force --install-extension twday.bibmanager
# nothing found for tikz
# metapost
$code --force --install-extension fjebaker.vscode-metapost
# gnuplot is separate below
# TBD: zotero

# gnuplot
$code --force --install-extension marioschwalbe.gnuplot
$code --force --install-extension fizzybreezy.gnuplot

# svg
$code --force --install-extension jock.svg
$code --force --install-extension simonsiefke.svg-preview

# spellchecker
code --force --install-extension valentjn.vscode-ltex

# perl (e.g. to configure latexmk)
$code --force --install-extension d9705996.perl-toolbox
```

Listing 3.2: Install script for extensions of VS Code.

```
# injection file written by latex plugin 2.0-SNAPSHOT
# config file for chktex
# adapted to latex-maven-plugin and to the according ant task

# LTeX: SETTINGS enabled=false

Silent {
  \textbackslash
}

WipeArg {
  \lstinputlisting:[]{} \lstset:{}
}

VerbEnvir
{
  Verbatim lstlisting
}

CmdLine {
  --nowarn 3 --nowarn 17
}

UserWarn={
}
```

Listing 3.3: The config file `.chktexrc` to check this manual.

`eissner/QMngMnt` uses the script for automation of installation and update.

For document development the tool `latexmk` is a valuable build tool. Also, a linter like `chktex` is helpful.

As described in [Col23], Section "CONFIGURATION/INITIALIZATION (RC) FILES", There are various configuration files `latexmkrc` or `.latexmkrc`, among these a global one, a local one referring to the enclosing folder and finally one specified by the command line option `-r`.

Likewise, [Thi22], Section 6.1.3, shows that also `chktexrc` has a global configuration file `chktexrc` and a local one `.chktexrc` or `chktexrc`, depending on the operating system. Finally, a configuration file can be specified with the option `-l`, according to [Thi22], Section 6.1.1. Unfortunately, the ordering in which the configuration file given by option is read in compared to the other, is not clearly specified.

For sake of reproducibility, we recommend restricting to the global configuration file which is tied to the installation and to a local file either respected because it is in the local folder or because it is given via a command line option.

Caution: According to [Thi22], Section 6.1.3, as described, the local configuration file fits only for UNIX-like operating systems. For Windows and that like, `chktexrc` is expected. This can be realized with a link. For sake of reproducibility, we recommend only global config files, a local one in the current directory and

maybe another one specified with the option `-r` as described below. In order to have the same `.chktexrc` for all main files in different folders, one shall use a central `.chktexrc`, e.g. `src/site/tex/.chktexrc`, and then either set a link to it from all folders with latex main files or just use the `-r`. Unfortunately, [Thi22] does not tell about the ordering in which the configuration file given by the option `-r` is read in.

This maven plugin offers a goal `inj` to create the configuration files `.latexmkrc`, `.chktexrc` and further files, all in the latex source directory. In particular `.latexmkrc` is adapted to the current settings of this plugin. That way, `latexmk` behaves the same as does this plugin. Similarly, for `chktex`. Note that these files are written only if either no file is overwritten or only files are overwritten which were written by this plugin. Self-written files are recognized by the headline. If this cannot be read or in some other exotic conditions, it cannot be ensured that the files are written by this software, and so they are not overwritten. In case of a doubt, a warning is displayed.

As mentioned above, the intention behind is, to keep `latexmk` and this latex plugin synchronized by providing properties in the pom and using them as settings when configuring the latex plugin and at the same time to filter the raw `.latexmkrc`. Currently, not all possible settings of this plugin are taken into account in the raw `.latexmkrc`. It is advisable, to use a raw `.latexmkrc` taking only parameters into account, which are explicitly configured for the latex plugin. So the given raw `.latexmkrc` is no more than a hint.

3.5 Development of documents

The term “development of documents” is coined by the author and reflects that writing a document resembles developing software in that it is an iterative process consisting in producing pieces of information, checking, modifying, correcting, erasing it, checking again…. After initial creation, is an iterative process like a dialog between the author and its work.

This is true of course independent of the tools used, but some tools support this process better than others. For document development the ideal are WYSIWYG (“what you see is what you get”) editors, which should maybe be better called WYRIWYR (“what you write is what you read”), or, taking also drawings into account, IisO (“input is output”). For software development the ideal languages are prototyping languages, interpreted at least.

From that point of view, LaTeX and friends is the worst conceivable choice:

- You write in an editor, but you read off from a viewer. So you must permanently switch your attention.

- You write a sequence of commands, but you read text, formulae, drawings. In a sense you program the appearance of a page or site.

  This discrepancy becomes particularly apparent if creating a drawing in LaTeX, e.g. with TikZ, because even drawings are described or programmed quite formally.

- You cannot just see instantly the result of your work; first you have to trigger a compilation process and wait some time. So, besides an editor and a viewer you also need some kind of console. It is even worse: Typically, based on the console output you must either rerun the compiler or run some auxiliary program, even more of them and then again the compiler, maybe several times. The decision whether the viewer shows the final result already, or whether another command has to be issued and if so which one, is based on the console output. So part of your attention must be on the console also. The console is also used to issue the next command.

- The compilation process may go wrong or be in a sense deficient, so what you need is observing logs, either on the console or in a log file. Even if the input is accepted by build tools even without warning, still there may be something wrong. The LaTeX tools do not include any spell checking or grammar checking. Since LaTeX documents are in a sense programmed, an additional burden is the need for a kind of linting, which is done, e.g. by `chktex`. This must be invoked manually and yields another log file, although no output.

The situation is visualized in Figure 3.1. It is no UML diagram although using elements of UML. The developer of the document is visualized as a stick figure and the tools used for development are the boxes surrounding it, resembling instances in a UML class diagram. Besides the tool under consideration, the according files are shown. The console is to invoke conversion commands like `lualatex`. This shows already, that the user does not face a single counterpart, but has to juggle with a bunch of tools at once. The arrows represent data flows and this data is commands at least partially, if the lines are solid, else they are dotted.

This explains the need for tools and techniques to mitigate the situation.

Seemingly, this LaTeX-builder is not to contribute to document development, because it is used after the end of the development process, automating the compilation process. But since compilation may fail and because it is also a checker tool, supervising even warnings, e.g. on bad boxes, and by default invoking `chktex` and monitoring its log file, it may be the start point for another loop in the development process.

Before describing the contribution of this LaTeX-builder to the process of document development, let us describe the process of document development in more

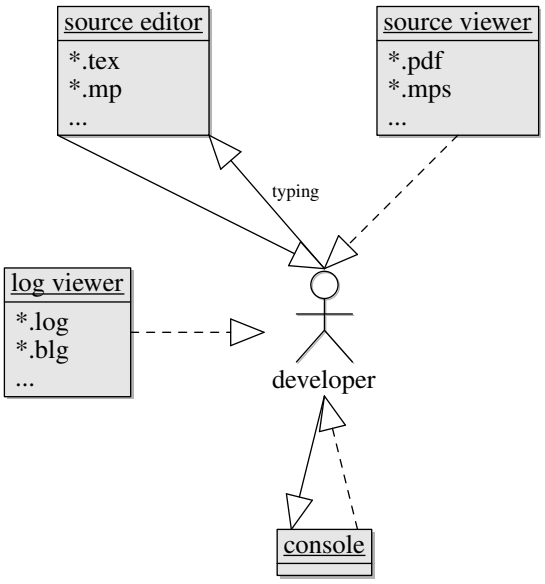


Figure 3.1: Document development with base tools

detail, in particular the other tools supporting document development and their interaction. With this background in mind, it is easy to describe the role of the LaTeX-builder in the team of development tools.

The minimum needed to develop a document in LaTeX are an editor, an according viewer and the LaTeX tools for build and check as described in Section 3.5.1. As described above, using this basic tools directly distracts much of the attention of the author/developer from the content. Thus, it is a good idea to use a tool to orchestrate the LaTeX tools. The author of this software prefers the orchestration tool `latexmk` which is described in Section 3.5.2.

The check tool `chktex` and the according goal `chk` are already described in Section 3.2. Nevertheless, the aspects of checking in the context of document development is treated separately in Section 3.5.3.

The goals `grp` and `clr` described in Sections 3.5.4 and 3.5.5 make sense only in the context of document development. For details see these sections.

Finally, Section 3.5.6 is on installing extensions for document development on the editor VS Code. To that end, this software provides an installation script.

3.5.1 Editors, viewers and LaTeX

The author recommends using VS Code to write LaTeX documents and to view the results on `okular`.

Editor VS Code with extensions in conjunction with viewer. Mention also

Emacs with AUCTEX. Point to installation script for extensions described in Section 3.5.6. Extensions of VS Code are mainly to highlight the code of the various file types, but the central extension is `james-yu.latex-workshop` which also provides build functionality.

For viewer, we use `okular`. Settings is settings configure okular. Then in tab `General` we

- deselect show backend selection dialog
- select reload document on file change

Describe forward search and backward search. Also mention clean goal described in more detail in Section 3.5.5

Mention a way to speed up compilation: using includes rather than imports. As a consequence, additional AUX files are created. These are also cleaned.

The latex processor may fail. Typically, the latex converter must be invoked more than once and besides the latex converter some further auxiliary programs must be run. What to do is displayed on the console. Most of the programs write success messages and more detailed information containing error messages, warning or just information messages in their respective log files. In the course of

Next talk about orchestration: The user is freed from deciding which of the many auxiliary programs are to be invoked next and whether the latex converter is to be invoked once more in order to emit final correct output.

There is another tool doing this work, `latexmk`. Essentially, it is better suited to document development, because it allows easily to build a single document only and is considerably faster because no overhead from maven or ant, and it can be run in `nonstopmode`.

Still, LaTeX-builder has its place in conjunction with `latexmk`.

3.5.2 The build tool `latexmk`

TBD: Describe the build tool `latexmk` but also its deficiencies, i.e. what this LaTeX-builder offers in addition. Essentially, there are two aspects: The build processes are different and the LaTeX-builder is much faster in conjunction with certain graphics as used e.g. in this manual. The main aspect is supervision of the build process: `latexmk` focuses on mere build success, but does not notify of warnings. So the build process may complete, but the result may be inappropriate.

Here also links to goal `grp` creating graphics files described in Section 3.5.4 and to Section 3.4 on file injection are in place. Also link on goal `clr` to clear created files described in Section 3.5.5.

3.5.3 Checks in the context of document development

TBD: rework

target and goal `chk` just invoke the tool `chktex` and checks for a finding. That way it is a quality control in the course of the build process, but also the entry point for further development.

As described in Section 3.4, the configuration file `.chktecrc` for `chktex` is injected.

In case of findings, a log file is written which can be analyzed. Then alternately the source can be corrected and `chktex` can be invoked directly. Since the configuration file is as in the build run, the results are the same.

3.5.4 Goal Graphics `grp`

Hint to relation with latexmk. needs mvn validate & mvn latex:grp.

For creating the graphic files in the TEX source directory, there is a goal *graphics*, invoked by `mvn latex:grp`. This goal does not create any output in the site directory. Instead, it populates the source directories with graphic files which can be directly included into the LaTeX-file and so it allows to run the LaTeX-compiler on the latex main files from within a development environment. Thus, the goal *graphics* is thus a vital feature for development of documents.

Note that in general `mvn clean validate latex:grp` creates all files necessary to compile with a latex converter like `lualatex` and also to compile smoothly with `latexmk`.

3.5.5 Goal Clear `clr`

Maybe the material is not enough for a separate section. Important to ensure independence.

Finally, there is another target for clearing the TEX source directory recursively, invoked by `mvn latex:clr`. For more details on the last three goals, see Section 3.5.

Hint to goal graphic.

As is described in more detail in Section 3.5, this software creates target documents and also intermediate files in the TEX source directory, at least with cleanup disabled. To eliminate the created files from the source directory, just type `mvn latex:clr`.

cleanup refers also to injections.

Likewise, goal `clr` deletes these configuration files if they were definitively written by this plugin. If this is proved to be false or a proof is not possible, the configuration files are not deleted. As for goal `inj`, in case of a doubt, a warning is displayed.

3.5.6 Installation and Configuration

TBD: rework: maybe better describe the goal `inj`. The goal `inj` is to create a set of files, partially adapted to the current configuration.

A first description of the injection goal is given by

```
mvn latex:help -Ddetail -Dgoal=inj
```

which yields a list of files which can be injected.

By default, it is tied to lifecycle phase `validate` and comprises the set of injections `latexmkrc,chktexrc`.

The first we treat is injection `vscodeExt` injecting a file `instVScode4tex.sh` in the TEX source directory. Typically, this is not injected during a lifecycle, but when installing or updating extensions for VS Code used during document development. Thus, typically it is invoked in the form

```
mvn latex:inj -Dlatex.injections=vscodeExt
```

In the default configuration, this creates an executable file

```
src/site/tex/instVScode4tex.sh
```

using bash shell. The extensions

Install script for installing extensions for VS Code helping in developing LaTeX documents.

In addition, configuration scripts for `latexmk` and `chktex`. Also describe how to use.

3.5.7 Miscellaneous

During development, it is comfortable, to have the log-file in the same directory as the LaTeX main file. Also, if PDF- and TEX-files are synchronized, also the PDF-file should be in the same directory. Likewise, files in graphic formats which cannot be included into a LaTeX-file without conversion, that converted file shall be in the same directory as the original one. So, all files, manually created files and files arising from automatic conversions shall be in the same folder, at least during development. Also, typically, one wants to mix creation by this maven-plugin or ant-task with at least partial creation through external tools. For example, if writing LaTeX-files with Emacs, it is much more convenient, to compile the LaTeX main file via `pdflatex` from within Emacs or to create a PDF-file from a fig-file through `xfig`'s export dialog, than using this maven-plugin or this ant-task. Also, these tools work best, if all is in one folder.

On the other hand, conventionally, in a maven project, sources are held in folder `src`, whereas created files occur in the folder `target`. Likewise for ant. The

compromise, this maven-plugin and this ant-task take, is, that at the end of a run, at most the files present at the beginning of the run may be present in the source directory. So, this software builds in the following steps:

- Store a list of all files present at the beginning of a run.
- Process all graphics files of the formats requiring preprocessing.
- Determine the LaTeX main files.
- Run the LaTeX converter, e.g. the one creating PDF-output or DOCX-output. This may include running auxiliary programs like `bibtex` or `pythontex` and also rerunning the LaTeX converter several times.
- Copy the result files (if any) into the target folder.
- Remove all files not present at the beginning of a run, by default.

To keep e.g. the resulting PDF, just create it via compilation through Emacs, even if not all graphic files to be included are present or just by a `touch`-command. Then in the next run of this plugin, this PDF will be re-created, that time complete with the graphics output. That way, synchronization between LaTeX- and PDF-files is possible. Likewise, to keep the log-file or the aux-file, just touch it. This technique is really valuable for debugging.

To keep all created files after a run of this maven-plugin, set the parameter `cleanUp` in the pom to `false` as illustrated in Listing 3.4. For the ant-task likewise.

But how can one get rid of all these newly created files? That is what is the goal `latex:clr` is for: `mvn latex:clr` removes all created graphic files and for each latex main file, it removes all files with "similar" names including log files, index files and that like. Typically, this suffices, to remove all files created. If not, try to modify parameter `$patternCreatedFromLatexMain` in the pom accordingly. If this does not help either, please inform the developer of this software. Of course, if further software is used which creates additional files, like Emacs creates a folder `auto`, these files cannot be removed by this maven-plugin or this ant-task. Note that `latex:clr` also removes exported files as listed in Section 3.2 from the target folder.

During development of a LaTeX-main file, it is often more convenient to compile from within an editor like Emacs. The problem is, that compilation fails if the graphic files are missing. This is what the goal *graphics* accessible via

```
mvn latex:grp
```

is for: It creates all graphic files required to compile the LaTeX-main files.

Still this does not create a bibliography, an index or a glossary. With *auctex*, an Emacs-package for editing LaTeX, bibliography and index are well-supported. To create a glossary, auctex has to be modified a little.

That way also the log-files required are created: In case of this manual, the files `manualLMP.xxx` are created where `xxx` is

- `log` for LaTeX,
- `blg` for `BibTeX`,
- `glg` for `makeglossaries` and
- `ilg` for `makeindex`.

The last goal regularly used for development of documentation is *check*. It is invoked via

```
mvn latex:chk
```

and runs `chktex`, described in [Thi22], on each latex main file after having created graphic files as for goal *graphics*. As a result, a log-file with suffix `.clg` is created but not copied to the target folder. If the log-file contains an entry, an according message is logged. Note that, with default configuration, `chktex` requires the LaTeX-package `booktabs` described in [Fea16].

Besides the basic configuration packaged with `chktex`, there can be an additional configuration file `.chktexrc` which partially overwrites variables set by the basic configuration file, partially, for list-valued variables, adds entries. Section 2.2 describes how to access the `.chktexrc` with which this manual is checked and details to the form of `.chktexrc` can be found in [Thi22], Section 6.1.5.

Finally, we have the goal `latex:vrs` to display meta information, above all version information:

```
mvn latex:vrs
```

displays something like what is displayed in Listing 3.5. Besides information on this software including version and even git commits, there are information on so-called registered converters, i.e. converters intended to be invoked by this software.

The goal yields a full list of registered converters, signifying which of them are excluded according to parameter `convertersExcluded`, which are not installed, and for each of the rest, the actual version, the allowed range and a warning if the actual version is out of range.

The parameter `convertersExcluded` is described in Table 6.1 on page 100. Excluded converters are prevented from being used: if tried, Exception TSS07 described in Table 7.4 on page 132 is thrown. If a converter is not installed, but tired

to be used, this kind of failure is obvious. Only if a converter is used with an unintended version bears some risk. Note that also unregistered converters can be used; but then the user is responsible to provide an appropriate version. An example for an unregistered converter is given in Table 6.8 on page 117: `pythontexW:pythontex` indicating the converter `pythontexW` with category `pythontex`.

As one can see, a warning WMI02 indicates that the version of a converter is out of the intended range, provided, the converter is installed, and it is not excluded according to the configuration `convertersExcluded`.

Note that in the given version and in the installation of the author, of course, all converters are installed and are up-to-date to be able to check validity. The according messages are forced for illustration only. For a user of this software which does no development, of course only converters need to be installed which are really needed.

Another aspect of document development is integration with other tools.

Document development starts with the editor. Above the Emacs editor enhanced with auctex was mentioned. We recommend VS Code in conjunction with several extensions. If VS Code itself is already installed the script in Listing 3.2 on page 33 installs and updates all extensions the author used to develop this manual. The core extension is `latex workshop`, the others are mainly used for editing graphic files. For details see Section 2.2.

A valuable standalone build tool is `latexmk`. It can even run in background. This manual can also be compiled with `latexmk` but only with adapted configuration file `.latexmkrc`. For details see Section 2.2.

This software and `latexmk` follow a different philosophy in finding dependencies: Whereas this software creates image files in advance before invoking a latex converter, `latexmk` first calls the latex converter in nonstopmode to avoid a stop because of a missing file. Then the file is created using the appropriate rule (hopefully unique) and the converter is run again, this time passing the inclusion of the first created files failing at the next one. This goes on that way until the last included file is created. Then the latex converter runs through without failure caused by missing files.

There are two problems with this: This yields a huge number of runs for the converter which is time-consuming and there is at least one kind of inclusion which does not work that way: inclusion with `\lstinputlisting` provided by `listings`. In fact, i have an email from J. Hoffmann, author of listings telling that there are more packages with the same problem. To be checked: `fancyvrb` and `moreverb`. Nevertheless, all other ways of inclusion *used by this manual* like the one with `\import` seem to work fine.

The current workaround is illustrated in Listing 3.6 modifying `listings`' output to make it digestible for `latexmk`.

Still some generalization in `latexmk` could spare this modification.

Another point is, that currently for each file `latexmk` creates with a separate rule, another run of the latex processor is required: The initial run is interrupted with the first missing file. Then that file is created by an appropriate rule and the latex processor is rerun failing with the next missing file. That way the process goes on until the last file is created with a rule. Of course this procedure is quite time-consuming, so an alternative is required.

3.6 Goals in the maven lifecycle

The goal `latex:cfg` exporting in the formats configured is tied to the lifecycle phase `site` so is invoked when commanding

```
mvn site
```

or subsequent phase.

Also, the goal `latex:clr` cleaning created files both from source directory and from target directory is tied to phase `clean` so is invoked when commanding

```
mvn clean
```

Finally, the goal `latex:vrs` displaying versions of converters and the goal `latex:inj` injecting a set of files depending on the configuration are tied to the phase `validate`. Thus, it is thus invoked when commanding

```
mvn validate
```

which is invoked not only in installation, but also by the site plugin. This ensures, that the converters are checked for correct version before being used. Note that by default, `mvn latex:vrs` displays complete version info, whereas `mvn validate` only displays warnings if appropriate. This is, because in the first case the plugin runs with the default `versionsWarnOnly=true` whereas in the second case, is configured with `versionsWarnOnly=false` as in Listing 2.4.

3.7 The ant-tasks

Section 3.2 treats goal `cfg` to create output from one source in various formats and also check which is without output. The target formats and also the checks are specified in the parameter `targets`.

There is an according ant task `cfg` doing the same also based on parameter `targets`. Whereas the maven plugin provides separate goals for each target, the ant-task has no such convenience feature. Section 3.2 briefly mentions goal `clr` used for cleanup. There is an according ant-task relying on according parameters.

Note that the ant task does not support very much of document development, but it is likely, that the user performs document development and runs other programs than the ant task on the sources. In this case, the `clr` task is vital.

If this ant-task is used in an ant project with folder structure conforming with a maven project and if the LaTeX sources do not require a special configuration, the above configuration is sufficient. Otherwise, parameters have to be given explicitly overwriting the default values.

```
<!-- create html and pdf and other formats from latex -->
<plugin>
  <groupId>eu.simuline.m2latex</groupId>
  <artifactId>latex-maven-plugin</artifactId>
  <version>2.0</version>

  <configuration>
    <settings>
      <targets>pdf</targets>
      <cleanUp>false</cleanUp>
    </settings>
  </configuration>
</plugin>
```

Listing 3.4: Configuration without cleanup

```
[INFO] --- latex:2.0-SNAPSHOT:vrs (default-cli) @ latex-maven-plugin ---
[INFO] Manifest properties:
[INFO] MANIFEST: (1.0)
[INFO]         Implementation-Version: '2.0-SNAPSHOT'
[INFO] PackageImplementation-Version: '2.0-SNAPSHOT'
[INFO] pom properties:
[INFO] coordinate.groupId:     'eu.simuline.m2latex'
[INFO] coordinate.artifactId: 'latex-maven-plugin'
[INFO] coordinate.version:     '2.0-SNAPSHOT'
[INFO] git properties:
[INFO] build version:  '2.0-SNAPSHOT'
[INFO] commit id desc: 'latex-maven-plugin-1.8-209-g5ac27b7-dirty'
[INFO] buildTime:       '2023-06-25T23:31:20+0200'
[INFO] tool versions:
[INFO] ?warn?     command              'actual version'(not)in[expected version interval]
[INFO]            pdflatex:            '1.40.25'in[1.40.21;1.40.25]
[INFO]            lualatex:            '1.17.0'in[1.12.0;1.17.0]
[INFO]            xelatex:             '0.999995'in[0.999992;0.999995]
[INFO]            latex2rtf:           '2.3.18 r1267'in[2.3.16 r1254;2.3.18 r1267]
[INFO]            odt2doc:             '0.9.0'in[0.9.0]
[INFO]            pdftotext:           '23.06.0'in[21.04.0;23.06.0]
[INFO]            dvips:               '2023.1'in[2020.1;2023.1]
[INFO]            dvipdfm:             '20220710'in[20210318;20220710]
[INFO]            dvipdfmx:            '20220710'in[20200315;20220710]
[INFO]            xdvipdfmx:           '20220710'in[20200315;20220710]
[INFO]            dvipdft:             '20090604.0046'in[20090604.0046]
[INFO]            gs:                  '9.56.1'in[9.52.0;9.56.1]
[INFO]            chktex:              '1.7.8'in[1.7.8]
[INFO]            diff-pdf-visually:   '1.7.0'in[1.6.4;1.7.0]
[INFO]            diff-pdf:            '300'in[300]
[INFO]            diff:                '3.10'in[3.8;3.10]
[INFO]            pdfinfo:             '23.06.0'in[22.01.0;23.06.0]
[INFO]            exiftool:            '12.63'in[12.39;12.63]
[INFO]            bibtex:              '0.99d'in[0.99d]
[INFO]            bibtexu:             '4.00'in[4.00;4.00]
[INFO]            bibtex8:             '4.00'in[4.00;4.00]
[WARNING] WM102:  makeindex:           '2.17'not in[2.15;2.16]
[INFO]            splitindex:          '0.1'in[0.1]
[INFO]            makeglossaries:      '4.51'in[4.45;4.51]
[INFO]            pythontex:           '0.18'in[0.17;0.18]
[INFO]            depythontex:         '0.18'in[0.17;0.18]
[INFO]            mpost:               '2.02'in[2.00;2.02]
[INFO]            ebb:                 '20220710'in[20200315;20220710]
[INFO]            gnuplot:             '5.4 patchlevel 8'in[5.4 patchlevel 0;5.4 patchlevel 8]
[INFO]            inkscape:            '1.2.2'in[1.0.2;1.2.2]
[INFO]            fig2dev:             '3.2.8b'in[3.2.7b;3.2.8b]
[INFO] tools excluded:
[INFO] upmendex, xindy
[INFO] tools not found:
[INFO] latexmk
[INFO] ------------------------------------------------------------------------
```

Listing 3.5: Output of goal `latex:vrs`

```
\usepackage{listings}
% this is a workaround for including listings with latexmk..
% This can be fixed
% - as shown below
% - patch in package listings
% - patch in latexmk
% I would prefer the latter.
\usepackage{xpatch}
\makeatletter
\newcommand*{\NewLine}{^^J}%
\xpatchcmd{\lst@MissingFileError}
{Package Listings Error: File `#1(.#2)' not found.}
{LaTeX Error: File `#1.#2' not found.\NewLine}{%
  \typeout{File ending patch for \string\lst@MissingFileError\space done.}%
}{%
  \typeout{File ending patch for \string\lst@MissingFileError\space failed.}%
}
\makeatother
```

Listing 3.6: Patch of the `listings` package given by the header file.

Chapter 4

Graphical Preprocessing

While LaTeX is really strong in text processing and also in formula processing, in itself it is weak in its graphical abilities. Graphics in some formats can be included directly, in some cases preprocessing is necessary to obtain includable graphics, but in most cases, according LaTeX-packages must be used:

graphicx is the basic graphics package which provides the command `\includegraphics` which allows including graphics natively in the formats pdf, eps, jpg and png at least. For details see [Car16].

xcolor allows using colors in graphics. Even if the author does not use colors in graphics, several formats, like fig, Gnuplot file format (gp) and svg offer it and so the according converters transforming them into the native formats create color information which can be rendered only via `xcolor`. For details see [Ker16].

transparent allows specifying transparency in graphics. Here the same applies as for color: Even if you do not use the feature, some source formats do (in fact only svg) does and so the according converters create according information and so LaTeX must get along with it. For details see [Obe16c].

bmpsize is needed for bitmap formats like jpg and png only. Used to extract resolution and bounding box. FIXME: needed more information. For details see [Obe16a].

tikz The tikz code described in [Tan15] is just in LaTeX format. Thus, it can be included directly anddoes not require any preprocessing. Still what is needed is a good graphical editor like `tikzedt` with online manual [TW12].

import is strictly speaking no graphics package. According to its documentation [Ars09], it allows an imported file to find its own inputs (using "`\input`",

"`\includegraphics`" etc.) in that directory. This is vital for the graphic formats for which a TEX file is imported which itself imports a PDF/EPS file located in the same folder but not in the folder of the importing file. It is advisable to combine the `import` package with other graphic packages to include graphics in separate graphic files.

pythontex is strictly speaking no graphics package either but more general a way to include and run code within a LaTeX document as described in [Poo21]. Note that not only Python but also other languages can be used. Most of them offer graphic capabilities and so graphics can be included also via `pythontex`. Nevertheless, we do not treat this technique in this chapter, but separately in Section 5.5. This is because graphics is a side aspect of `pythontex` and also because strictly speaking there is no preprocessing. First a latex processor is run, and the package extracts the code into a separate file which is then further processed by an external tool. This is more like running `\bibtex` to extract a bibliography.

If using the package `pythontex` a special processing interacting with the latex to PDF converter is required also, but it is not preprocessing.

We support figures created in the fig by the program `xfig`, figures created by the plotting utility `gnuplot`, the TEX specific graphic language MetaPost, figures in svg-format and finally figures in formats which can be included into a LaTeX-file without preprocessing like png and jpg. Further functionality on LaTeX figures can be easily added. If there is some need, please write an email to the author.

Table 4.1 gives an overview over the supported formats together with the graphic converters, the name of the parameter to configure which one is used and the default value. Of course this converter must be installed. It is advisable to have also an editor installed. Sometimes the editor is used also as converter. For human-readable formats like `fig`, it often makes sense, to use both the graphical editor and the textual one.

Graphic format	conversion tool	editor
fig	`fig2devCommand`, e.g. `fig2dev`	xfig, emacs
gnuplot (gp)	`gnuplotCommand`, e.g. `gnuplot`	emacs
MetaPost (mp)	`metapostCommand`, e.g. `mpost`	emacs
svg	`svg2devCommand`, e.g. `inkscape`	inkscape
jpg, png	—	gimp

Table 4.1: Overview over the supported graphic formats

Besides the converter external to LaTeX, also several LaTeX-packages are required to use graphics.

This section describes the conversions of graphical source files into target files in detail.

But PDF also occurs as an intermediate format for pictures. For historical reasons, still eps is used. Section 4.1 shows that it suffices to stick to pictures in PDF format. Section 4.2 shows how `fig2dev` converts fig-files into LaTeX-files containing text and including graphics in as PDF files. Likewise, Section 4.3 describes how gnuplot converts gnuplot-files into PDF files. An interesting alternative to gnuplot for computing pictures is MetaPost described in Section 4.4. A more elaborate alternative to fig-pictures are SVG pictures described in Section 4.5 Also several formats collected in Section 4.6 may be included as is.

4.1 Inclusion of PDF files

Modern LaTeX implementations directly create PDF files. Thus, it makes sense, to allow including graphics as PDF files in LaTeX-files. This is done in PDF mode by the LaTeX-packages `xcolor` and `graphicx`. In contrast, traditionally LaTeX produced output in the dvi-format which is still used to create html-output, this is not supported by the default driver. Instead, it shall be used the driver `dvipdfmx`. To obtain a LaTeX-file which works both for creating PDF and HTML, insert in the header of the LaTeX main file given by Listing 4.1. Note that also the package `hyperref` needs adaption of its options depending on the use case.

An alternative, which we do not use is replacing the pdf by an eps via configuration file `myxhtml.cfg` which is given by Listing 4.2. It applies when converting LaTeX into html and that like with htlatex if invoking something like `htlatex xxx.tex myxhtml,…` where `myxhtml` is located in the same directory as the LaTeX-file to be processed. The configuration file essentially contains a graphics configuration, which applies conversion to pdf-files included in the LaTeX-file `xxx.tex` via `\includegraphics`.

4.2 Conversion of fig-files

A simple but still useful tool to draw figures is `xfig` which stores graphics in a native format described in [Rei16] with file extension `.fig`. The file extension `.fig` is also used by MATLAB to store plots, but this is something different. Graphics in xfig format cannot be directly included in latex files but must be exported into a LaTeX-readable format.

To export a file `xxx.fig` residing in directory `yyy` into several external formats, `xfig` uses `fig2dev`. A look in [Rei16], Section 3.4 shows that texts with set "special"-flag are interpreted as latex-code. For these texts the appropriate export

```
\ifpdf%
  % for accessability with luatex
  %\usepackage{luatex85}
  % compiles for xelatex only
  %\usepackage[tagged, highstructure]{accessibility}
  \usepackage{xcolor}  % [pdftex]
  \usepackage{graphicx}% [pdftex]
  % driver [hpdftex] is autodetected
  \usepackage[destlabel]{hyperref}
  % sometimes comes in with svg import
  \usepackage{transparent}
  % warning transparent package:
  % loading aborted if not pdf-mode
  % strange: according to documentation not for xelatex;
  % seems to work anyway
  % can be extended using l3opacity
\else
  % No PDF, includes dvi/xdv and HTML,... via package tex4ht
  \usepackage[dvipdfmx]{xcolor}
  \usepackage[dvipdfmx]{graphicx}
  \IfPackageLoadedTF{tex4ht}{%
    \usepackage[tex4ht, destlabel]{hyperref}
  }{%
    \ifxetex%
      \usepackage[destlabel]{hyperref}
    \else
      \usepackage[dvipdfmx, destlabel]{hyperref}%[dvipdfmx]
      % lualatex: without [dvipdfmx] option did not find
      % converter dvi to pdf or to ps
      % pdflatex: without [dvipdfmx] option dvips still works,
      % but no converter for pdf
    \fi
  }% tex4ht not loaded
  %\usepackage{bmpsize}% not for xelatex
\fi%ifpdf
```

Listing 4.1: Loading packages adapted to the output format

```
\Preamble{xhtml}
%\makeatletter
\Configure{graphics*}
  {pdf}
  {
    \Needs{"convert       \csname Gin@base\endcsname.pdf %
                          \csname Gin@base\endcsname.eps"}%
    \Picture[picture not present]       {\csname Gin@base\endcsname.eps}%
    \special{t4ht+@File: \csname Gin@base\endcsname.eps}
  }
\begin{document}
\EndPreamble
\input{myxhtml.cfg}
```

Listing 4.2: Configuration file `myxhtml.cfg`

language would be `latex`. On the other hand, `latex` is weak in graphics and `pdf` would be the ideal export format for all kinds of objects, except for texts with set "special"-flag. In `pdf` format, texts are interpreted literally, independent of the "special"-flag. Thus, `fig2dev` offers a mixed solution: export `xxx.fig` in format `pdftex` which yields a pdf-file `xxx.pdf` containing all but text with set "special"-flag and complementary `pdftex_t` which yields a tex-file `xxx.ptx` including the pdf-file and the texts with set "special"-flag. The exported files are in the same directory `yyy` as the original file `xxx.fig`.

For example, the fig-file `F4_01fig2dev.fig` defining Figure 4.1, is transformed into a file `F4_01fig2dev.ptx` in format `pdftex_t` which starts as given by Listing 4.3.

The file `xxx.ptx` is "imported" into the tex-file of this manual by the command

```
\import{yyy}{xxx.ptx}
```

and includes `xxx.pdf` automatically the file `xxx.pdf` via `\includegraphics{xxx}` (line 2). Note the following remarkable details:

- Observe that we can drop the suffix of the included file `xxx.pdf` which is expressed as "`xxx`" because LaTeX chooses the right suffix: If instead of `xxx.pdf` there is a file `xxx.eps`, the latter is chosen if no suffix is specified. As we will see below, omitting the suffix is crucial to make `xxx.ptx` work for both LaTeX-output formats: the pdf-format can include pdf-files, whereas the dvi-format which is required to create html- and odt-files can include eps-files.

- If `xxx.pdf` is included in `xxx.ptx` with the full path name, we may use `\input{xxx.ptx}` instead of `\import{yyy}{xxx.ptx}`.

  If in contrast, `xxx.pdf` is included in `xxx.ptx` with the short name only, `xxx.pdf` is assumed to be in the same directory as the file inputting `xxx.ptx`.

```
\begin{picture}(0,0)%
\includegraphics{F4_01fig2dev}%
\end{picture}%
%
%  Conversion of xxx.fig into xxx.ptx, xxx.pdf and xxx.eps
%
\setlength{\unitlength}{2072sp}%
\begin{picture}(8492,4797)(1114,-4621)
\put(1351,-2311){\makebox(0,0)[lb]{\smash{\fontsize{10}{12}\usefont{T1}{ptm}{m}{n}{\color[rgb]{0,0,0}\texttt{xxx.fig
    }}%
}}}
\put(6976,-286){\makebox(0,0)[lb]{\smash{\fontsize{10}{12}\usefont{T1}{ptm}{m}{n}{\color[rgb]{0,0,0}\texttt{xxx.pdf
    }}%
}}}
\put(6976,-2311){\makebox(0,0)[lb]{\smash{\fontsize{10}{12}\usefont{T1}{ptm}{m}{n}{\color[rgb]{0,0,0}\texttt{xxx.ptx
    }}%
}}}
\put(6976,-4336){\makebox(0,0)[lb]{\smash{\fontsize{10}{12}\usefont{T1}{ptm}{m}{n}{\color[rgb]{0,0,0}\texttt{xxx.eps
    }}%
}}}
\put(4726,-1636){\makebox(0,0)[b]{\smash{\fontsize{10}{12}\usefont{T1}{ptm}{m}{n}{\color[rgb]{0,0,0}\texttt{fig2dev
    -L pdftex\_t}}%
}}}
\put(4726,-2311){\makebox(0,0)[b]{\smash{\fontsize{10}{12}\usefont{T1}{ptm}{m}{n}{\color[rgb]{0,0,0}\texttt{fig2dev
    -L pstex\_t}}%
}}}
\put(3826,-61){\makebox(0,0)[rb]{\smash{\fontsize{10}{12}\usefont{T1}{ptm}{m}{n}{\color[rgb]{0,0,0}\texttt{fig2dev -
    L pdftex}}%
}}}
\put(3826,-4561){\makebox(0,0)[rb]{\smash{\fontsize{10}{12}\usefont{T1}{ptm}{m}{n}{\color[rgb]{0,0,0}\texttt{fig2dev
     -L pstex}}%
}}}
\put(7426,-1186){\makebox(0,0)[lb]{\smash{\fontsize{10}{12}\usefont{T1}{ptm}{m}{n}{\color[rgb]{0,0,0}\texttt{\
    textbackslash includegraphics\{xxx\}}}%
```

Listing 4.3: The ptx-file for a fig-file

> So in general, i.e. if this is not `yyy`, we need import `\import{yyy}{xxx.ptx}`. If the directories coincide, in the import the string `yyy` may be empty. If the string `yyy` is not empty, it must end with the path delimiter, i.e. `/` for Unix like systems and \ for win-like systems.

As indicated in Section 4.1, the commands in `xxx.ptx` require the packages `graphicx` and `xcolor`. Also the `\import` command requires the `import` package.

To export `xxx.fig` into `xxx.ptx` and `xxx.pdf` this software invokes two commands:

```
fig2dev -L pdftex   <fig2devGenOptions> <fig2devPdfEpsOptions>        xxx.fig xxx.pdf
fig2dev -L pdftex_t <fig2devGenOptions> <fig2devPtxOptions>    -p xxx xxx.fig xxx.ptx
```

Both commands specify the input file `xxx.fig`, both use the options given by the parameter `fig2devGenOptions` while each invocation allows to specify also specific options, `fig2devPdfEpsOptions` and `fig2devPtxOptions`, respectively, and both use the option `-L` to specify the output format ("language").

The parameters specific for `pdftex` are called `fig2devPdfEpsOptions` because the options available are the same as for output format `pstex` creating eps-files. An example for a common option would be `-b width` which shall specify the same boundary for both formats; otherwise they do not fit.

For the output format `pdftex_t`, the option `-p xxx` says, that the string `xxx` must be included in `xxx.ptx` as `\includegraphics{xxx}`. Note that the option

`-p` shall not be specified in `fig2devPtxOptions`, because it is automatically added.

Equivalent to mixed export with formats `pdftex` and `pdftex_t` which is appropriate for LaTeX-output format pdf, is the mixed export with the according formats `pstex` and `pstex_t` appropriate for LaTeX-output format dvi. The difference is that `pstex` creates an eps-file instead of a pdf-file with the same content and `pstex_t` creates a tex-file which looks like that created by `pdftex_t` except including the eps-file instead of the pdf-file. If the suffix is not given, `pstex_t` and `pdftex_t` create identical files. Thus exporting `xxx.fig` via

```
fig2dev -L pstex    <fig2devGenOptions> <fig2devPdfEpsOptions>        xxx.fig xxx.eps
fig2dev -L pdftex   <fig2devGenOptions> <fig2devPdfEpsOptions>        xxx.fig xxx.pdf
fig2dev -L pdftex_t <fig2devGenOptions> <fig2devPtxOptions>    -p xxx xxx.fig xxx.ptx
```

and "inputting" `xxx.ptx` works for both LaTeX output formats.

Table 4.2 relates the language specified with the `-L` option with the suffix of the output file chosen canonically, the suffix we choose and the actual file format. In contrast to `fig2dev`, we choose the actual file format, except if this is TEX. We opted for the quite unusual suffix `.ptx` instead of `.tex` to avoid that TEX-files may be both, source files and created files, but this is not compulsory, since the same holds and is accepted for pdf-files.

Output format (language)	xfig suffix	our suffix	format
pstex	pstex	eps	eps
pstex_t	pstex_t	ptx	tex
pdftex	pdf	pdf	pdf
pdftex_t	pdf_t	pdf	pdf

Table 4.2: Language, suffixes and file format

Maybe xfig is intended to export from within the export dialog and not directly via a script like `fig2dev`. This may be the reason why the magnification must be set in the export dialog, but it is stored in the fig-file nevertheless.

Figure 4.1 shows the transformation of figures with `fig2dev` and the inclusion of the eps-file and of the pdf-file in the ptx-file. Note that the `fig2dev`-command is configurable via the parameter `fig2devCommand`, but there will be hardly any command with the same command line interface performing exactly the transformations given in Figure 4.1, except `fig2dev` itself.

At the same time, Figure 4.1 is an example for a LaTeX-file `xxx.ptx` created from a fig-file and embedded in this LaTeX-file with the `\input`-command. More than that, Figure 4.1 describes the way it has been created. Note that all text labels are specified with set "special"-flag, and are thus included as LaTeX-text, except the text postscript which is typeset with a postscript font to make the difference visible.

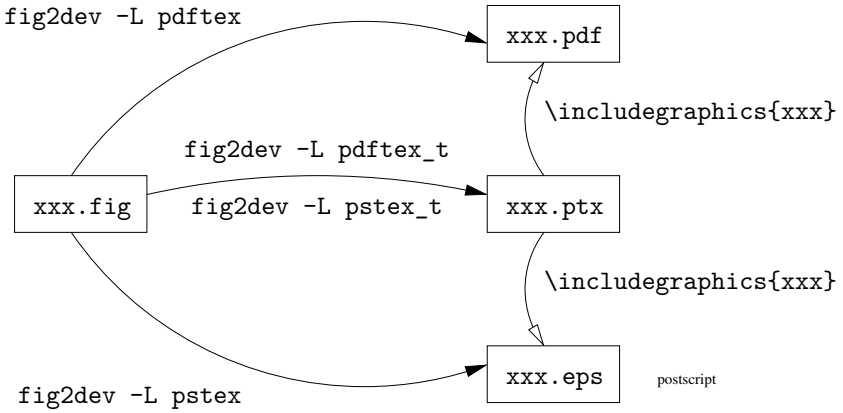


Figure 4.1: Conversion of a fig-file into pdf-, eps- and ptx-files with inclusions

4.3 Conversion of gnuplot-files

The term "gnuplot" refers to a file format and to a program `gnuplot` which can read this format, both described in [WK20].

Note that there seems no official file extension to identify gnuplot files. From the most common extensions `.plt`, `.gpi` and `.gp` we have chosen the one with the least collision and supported by Emacs, vscode and by my file browser: `.gp`.

The gnuplot format is a textual command language you can even program with and may thus be created with any editor but for sake of reproducibility it is recommended to use only files created by `gnuplot`. To ensure that a hand-written gnuplot file `xxx.gp`, e.g. with a single line like

```
plot [-10:10] sin(x), atan(x), cos(atan(x))
```

really works with the current `gnuplot` and to see how it is interpreted, it is recommended to convert it via

```
gnuplot -persist -e "load 'xxx.gp'; save 'xxx.gp'"
```

If you have a look inside the resulting file `F4_03someGnuplot.gp`, you can see, that in a comment line the current version of `gnuplot` is documented and also all the settings implicitly used. The original line is the last but one. Pasting the into vscode one can see the highlighting, of course provided the extensions Section 3.5 and given by Listing 3.2 are installed.

Also if a gnuplot file is created with an old version of `gnuplot`, it is recommended to update version with the same command. Note that `gnuplot` does not offer full backward compatibility.

This software supports including figures stored in `.gp`-files created by `gnuplot`. To export a file `xxx.gp` into several external formats, it uses `gnuplot` itself. According to the manual [WK20], Part IV, `gnuplot` supports output formats through

so called *terminals*. Among those are several ones intended for inclusion into LaTeX-files, like `Cairolatex`, `Eepic`, `Epslatex`, `Latex`, `Lua`, `Mf`, `Mp`, `Postscript`, `Ps(la)tex`, `Pstricks`, `Texdraw` and `Tpic`. Note that also export into the fig-format via the terminal `Fig` is supported which in turn may be included in latex as described in Section 4.2. Also `gnuplot` pictures may be exported in MetaPost format which in turn may be included in latex as described in Section 4.4.

This software supports the export of a file `xxx.gp` only via the terminal `Cairolatex` which offers export to mixed pdf and LaTeX: graphics in pdf and text in LaTeX which yields the fonts typical for LaTeX. This is as described for fig-files in Section 4.2, except that text is generally converted in LaTeX-format, and not selectively those text marked with special flag.

Accordingly, the export yields two files `xxx.ptx` and `xxx.pdf`, both in the directory `yyy` in which `xxx.gp` resides. The file `xxx.ptx` must be imported via

```
\import{yyy}{xxx.ptx}
```

It contains the texts and includes `xxx.pdf` via `\includegraphics{xxx}` without specifying a suffix.

Unlike for fig-files, `xxx.ptx` and `xxx.pdf` are created with a single command:

```
gnuplot -e "set terminal cairolatex pdf <gnuplotOptions>;
            set output 'xxx.ptx';
            load 'xxx.gp'"
```

Accordingly, `xxx.ptx` and `xxx.eps` are created with a single command:

```
gnuplot -e "set terminal cairolatex eps <gnuplotOptions>;
            set output 'xxx.ptx';
            load 'xxx.gp'"
```

Note that this writes another but identical file `xxx.ptx` as no file endings are written and so `xxx.ptx` can include both, `pdf` and `eps`. When creating both performance is not optimal, but `gnuplot` offers no way to avoid this. If being strict, `xxx.ptx` is perfectly correct only for output `eps`, if comments and error messages are taken into account but as long as no error occurs, the result is perfectly ok also for `pdf`.

As for inclusion of fig-files, packages `graphicx` and `color` are needed.

Figure 4.2 shows the transformation of the plots and the inclusion of graphic files. In addition, Figure 4.3 shows an example of a LaTeX-file created from a gnuplot file and embedded in this LaTeX-file.

4.4 Inclusion of MetaPost files

A vector graphic format, very native to TeX is `MetaPost`, a derivative of `Metafont` originally used to describe shape of fonts. Although seemingly supported by TeX

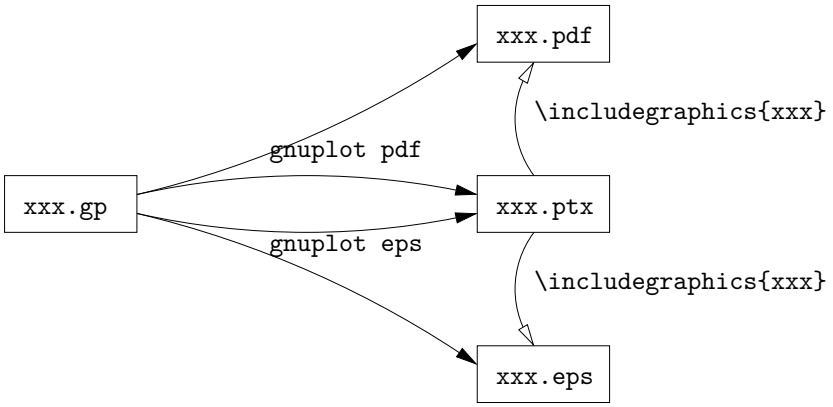


Figure 4.2: Conversion of a gnuplot-file into pdf-, eps- and ptx-files with inclusions

only, `MetaPost` is interesting in its own right, as it is a graphical programming language, Turing complete, much like postscript, and allows also declarative programming. The manual describing the language is [Hob14], seemingly complete, but it is not. Thus, one can be thankful for [HH13] which offers some introduction and for the really helpful tutorial [Hec05].

Files containing `MetaPost` have the ending `.mp`. Note that there are other graphic formats like monochrome pictures in TIFF-format which are identified with the same extension but the MetaPost format has nothing to do with this.

Since `MetaPost` is a programming language, MetaPost files are created with an editor. Since `MetaPost` is very versatile, it is impossible to give an impression by a single example. We decided to choose an example using a MetaPost library, `MetaUML`, described in [Ghe19] for some reasons apparent later. The example file is given in Listing 4.4 and also on the web as `F4_05someMetapost.mp`. It is the source file of Figure 4.5. Pasting the into VS Code, one can see the highlighting, of course provided the extensions Section 3.5 and given by Listing 3.2 are installed.

Listing 4.4 illustrates some structure of MetaPost. As in TEX, comments start with `%` and end with the line or with the file. The proper figures are enclosed between `beginfig(n)` and `endfig`, where n is the number of the figure, the so called `charcode`[1], and the file ends with `end`. This software relies on specifying a single figure per file; the `charcode` is irrelevant.

Code outside figures is possible, but does not belong to a figure and is thus not displayed. In our example, besides `end` commands outside the figure are just `input xxx`, where `xxx` names a so-called library defined by the file `xxx.mp` and a sequence of settings of internal variables of the MetaPost compiler controlling how the following figure is compiled. Most of them even in comments.

[1]This is a relict from Metafont, where each figure showed a character

```
%prologues := 0;% default
%%prologues := 3;
%outputtemplate:="%{jobname}.%{charcode}";% default
%outputtemplate:="%{jobname}%{charcode}.mps";% for latex
%outputformat:="eps";% default
input metauml;
beginfig(1);
  % states
  Begin.beginAll;
  %End.endAll;
  % State Standby
  State.Stopped("STOPPED")();
  Stopped.w = beginAll.e + (20, 0);
  State.Playing("PLAYING")();
  %Playing.w = Stopped.e + (60, 0);
  State.Paused("PAUSED")();
  Stopped.n = 0.5[Paused.n, Playing.n] + (0, 70);
  Playing.w = Paused.e + (120, 0);
  %endAll.w  = Standby.e  + (20,0);

  Note.A("This is my aleph", btex $\aleph$ etex);
  A.e = beginAll.w + (-20, 0);

  drawObjects(beginAll, Stopped, Playing, Paused, A);

  % feedback; links after draw: bad

  % links between states
  link(transition)(beginAll.e -- Stopped.w);
  link(transition)(Stopped.e  -- Playing.n);
  item.play(iAssoc)("[play()]")
    (play.sw = 0.5[Stopped.e, Playing.n]);
  link(transition)(Playing.nw  -- Stopped.se);
  item.stopPlaying(iAssoc)("[stop()]")
    (stopPlaying.ne = 0.5[Playing.nw, Stopped.se]);
  link(transition)(Playing.w + (0, +10) -- Paused.e  + (0, +10));
  item.pause(iAssoc)("[pause()]")
    (pause.s = 0.5[Playing.w, Paused.e] + (0, 10));
  link(transition)(Paused.e  + (0, -10) -- Playing.w + (0, -10));
  item.playPaused(iAssoc)("[play()]")
    (playPaused.n = 0.5[Paused.e, Playing.w] + (0, -10));
  link(transition)(Paused.ne -- Stopped.sw);
  item.stopPaused(iAssoc)("[stop()]")
    (stopPaused.nw = 0.5[Paused.ne, Stopped.sw]);
endfig;
end
```

Listing 4.4: An example file in MetaPost

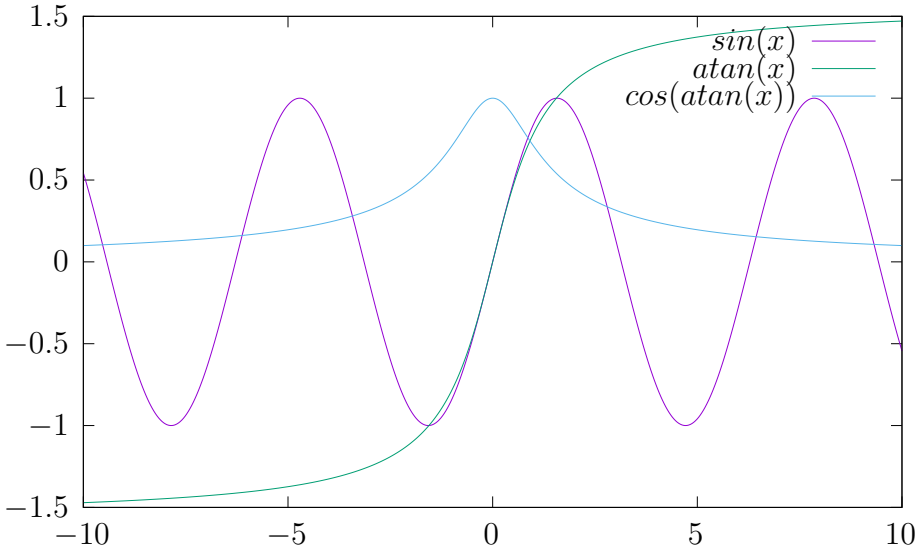


Figure 4.3: Converted sample gnuplot-file into ptx and pdf files

The compiler for `MetaPost` is given by the parameter `metapostCommand` which defaults to `mpost`, occasionally just `mp`.

Each internal variable which can be set in the MP file can also be set when invoking `mpost` using the option `-s ⟨variable⟩=⟨value⟩` as described in [Hob14], Section B.2.1. There it is stated that the option is read just before the file is read, which implies that the setting in the file overrides the command line setting.

The most basic setting is `outputformat:="eps"` which is the only setting appropriate for latex. So don't change[2]. Note the strange default setting for the names of the output files, `outputtemplate`, which reflects the `charcode` of the individual figures as file ending. For inclusion in latex, the file ending `mps` is required and so frequently `outputtemplate` is set to reflect the ending. It seems more appropriate to make the setting in the command line which yields the following invocation

```
mpost -s 'outputtemplate="%{jobname}%{charcode}.mps"' xxx.mp
```

As we agreed that a MetaPost file shall contain a single figure only, we also ignore the `charcode` which unifies MetaPost with other formats supported. This yields

```
mpost -s 'outputtemplate="%{jobname}.mps"' xxx.mp
```

[2]Note that `metapostCommand` may also besides eps output svg and png, just by setting `outputformat:="svg"` or that like. Caution: case-sensitive, assuming silently `eps` if the format is not recognized. Whereas svg is a vector format as MetaPost ifself, png is a raster format

The MetaPost file shall not overwrite the command line settings.

The setting of `prologues` controls where fonts come from and becomes relevant when using TEX for typesetting. Listing 4.4, line 21 includes a label via a note implicitly, and for the material between `btex` and `etex` uses TEX. The manual [Hob14], Section 8.1 is on typesetting labels and specifies the meaning of `prologues`. If we stick to including in LATEX and creating PDF out of that only, the default setting 0 is appropriate always but since this software uses DVI as intermediate format, e.g. to create HTML, or because for debugging one wants to view the MPS files standalone in a viewer things are not so easy. For details see [Hob14], Section 14.2. Setting `prologues:=1` is deprecated. The only save way to get the correct display is to include fonts in the MPS file, setting `prologues:=3`, but this makes the MPS file quite big. So a good compromise is to set `prologues:=2` as a command line option resulting in

```
mpost -s prologues=2 -s 'outputtemplate="%{jobname}.mps"' xxx.mp
```

and overwriting by need as in Listing 4.4, line 2.

As mentioned above, `input xxx` includes a library making the program dependent on a file `xxx.mp`. As for latex processors, also `mpost` records dependencies recursively in an FLS file if invoked with option `-recorder`. Also like latex processors, an error shall not cause break or interaction so adding the option `-interaction=nonstopmode`. Thus, we arrive finally at the default invocation

```
mpost -interaction=nonstopmode -recorder \
  -s prologues=2 -s 'outputtemplate="%{jobname}.mps"' xxx.mp
```

Figure 4.4 illustrates how `mpost` converts an mp-file `xxx.mp` with the given settings into various result files:

- an mps-file or with setting

  ```
  outputtemplate="\%{jobname}\%{charcode}.mps"
  ```

  more mps-files `xxx1.mps`…`xxxn.mps`,
- a log-file `xxx.log` and a fls-file `xxx.fls` much like LATEX does
- and an metapost tex output: texts (mpx)-file `xxx.mpx` containing the LATEX text of the figure; this is not created if there is no such text.

Figure 4.5 gives an example of a MetaPost file included in this LATEX-file as ab mps-file created from the MetaPost file and embedded in this LATEX-file with the `\includegraphics`-command. Normally, `\includegraphics` is invoked with

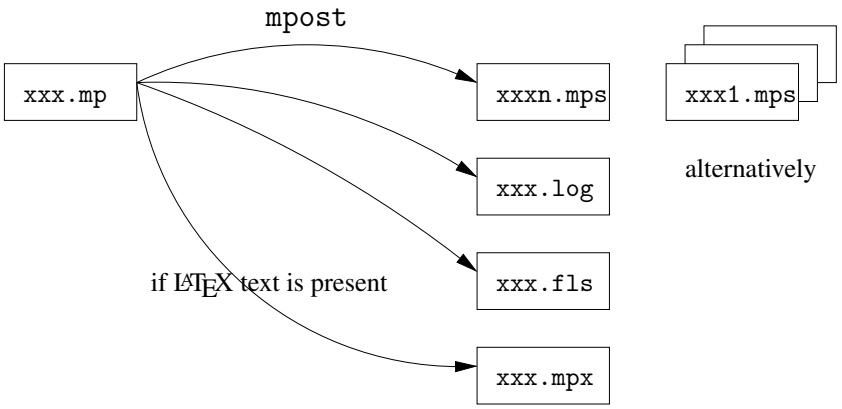


Figure 4.4: Conversion of a MetaPost-file into an mps-file

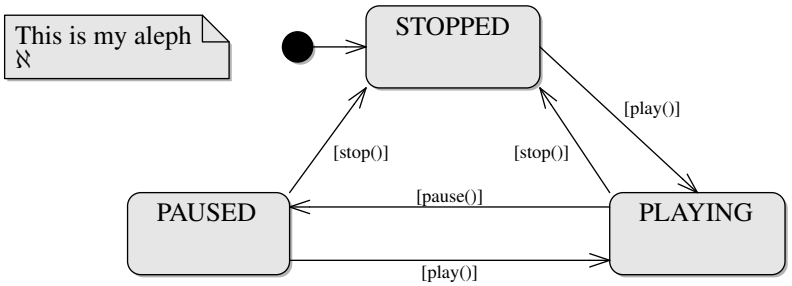


Figure 4.5: Converted sample MetaPost-file included as mps-file

the filename without extension, but for mps-files, the extension is needed. As for inclusion of fig-files, the package `graphicx` is needed.

One of the descendants of MetaPost is TikZ (see introductory text [Cré11]) and one of the deficiencies resolved is that it allows passing information from the main document to the proper figure.

With `lualatex` this can be reached for MetaPost also using package `luamplib`

The package itself provides an environment `mplibcode`. Essentially, `lualatex` interprets all code enclosed in the `mplibcode` environment as MetaPost. The package is enhanced with the command `\inputmpcode` which allows also load MetaPost from a file. The latter is preferred to direct inclusion with the `mplibcode`

```
\ifluatex%
  \usepackage{luamplib}
  \newcommand*\inputmpcode[1]{\begin{mplibcode}input #1\end{mplibcode}}
\else
\fi
```

Listing 4.5: Package `luamplib` and how to use it

environment, e.g. for sake of proper code highlighting. Note that the package declaration is enclosed in an if construct, ensuring that the package is loaded only if lualatex or that like is run.

That this allows better integration within the enclosing latex document is illustrated by redefining the letter ℵ as α which is really related.

```
{% make redefine local
\renewcommand{\aleph}{\alpha}
\inputtmpcode{F4_05someMetapost}
}% to recover from redefine {manualC4graphics.tex}
```

Figure 4.6 Documents, that the redefinition really influences rendering in the MetaPost file.

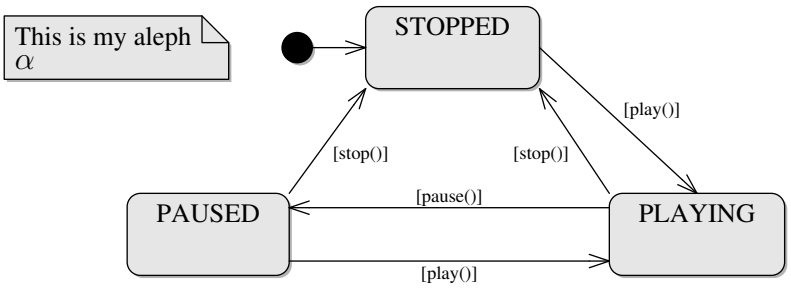


Figure 4.6: Sample MetaPost-file included via `luamplib` for lua(hb)tex

4.5 Inclusion of svg-files

Comparable with the xfig-format described in Section 4.2 but much more elaborate and widely used is the svg-format. There is a huge up-to-date official SVG 1.1 specification, [Da11] and a specification [Aa08] for SVG Tiny 1.2, which is itself quite short and more readable and gives also a good overview on "SVG Big". For a tutorial, see [DHH02]. As stated in [Aa08], Section 1.1, svg-files may contain vector graphics, raster images and text. It may also contain video and audio elements and may be interactive and dynamic, which goes beyond what can be included in LaTeX-files.

Figure 4.8 shows a picture in svg-format. As pdf-files are included directly via the `\includegraphics`-command, using the LaTeX-packages `xcolor` and `graphicx`, virtually, `xxx.svg` can be included directly via

```
%\includesvg[width=0.5\textwidth]{xxx}%
```

using the LaTeX-packages `svg` described in [Ilt12]. Note that the suffix of the file name shall be omitted.

A closer look shows, that graphic preprocessing is done behind the scenes in the course of a LaTeX-run creating files `xxx.pdf` and `xxx.pdf_tex`. As described for fig-files in Section 4.2 and for gnuplot-files in Section 4.3: The latter is a LaTeX-file containing text and including the former. To include `xxx.pdf` of course the LaTeX-packages `xcolor` and `graphicx` are required. Moreover, it may happen that the LaTeX-package `transparent` is required also, depending on the features used in `xxx.svg`.

As indicated in [Ilt12], Section 1, the `svg`-package delegates the transformation of `xxx.svg` `xxx.pdf` and `xxx.pdf_tex` to `inkscape`. This is a graphical editor with export functions which can be invoked in batch-mode also. Of course using the `svg`-package has the advantage that no explicit preprocessing is required, the created files updated by need. It is worth thinking about whether it is worthwhile writing according packages `fig` and `gnuplot`.

On the other hand, this breaks the workflow this software normally applies to graphic files. In particular, the package creates latex main files which are not removed after the latex run if parametrized accordingly or if something goes wrong. Also, the `svg`-package does not provide the full flexibility of a standard solution. Since this software is still under construction and more than that, is in an experimental phase, we provide explicit preprocessing of svg-files using `inkscape`. Another problem with the `svg`-package is, that according to [Ilt12], Section 1, it does not work on Windows platforms.

Some research shows, that `inkscape` in the version current at time of this writing exports mixed PDF and latex: If invoked as

```
inkscape --export-filename=xxx.pdf --export-area-drawing --export-latex xxx.svg
```

`inkscape` creates a file `xxx.pdf` containing all graphics but text and another file `xxx.pdf_tex` containing text and including `xxx.pdf`. The file `xxx.pdf_tex` can be be integrated into the latex document as

```
\def\svgwidth{0.5\textwidth}
\import{yyy}{xxx.pdf\_tex}%
```

Unlike `fig2dev` and `gnuplot`, specifying the files with their full path, has no effect, i.e. inclusion uses the file name only. Thus `\import` cannot be replaced by `\input` and so the LaTeX-package `import` is required.

This is essentially the same technique as applied for fig-files and for gnuplot-files as described in Sections 4.2 and 4.3.

Analogously,

```
inkscape --export-filename=xxx.eps --export-area-drawing --export-latex xxx.svg
```

exports files `xxx.eps_tex` and `xxx.eps`.

In older versions of `inkscape`, there was a configuration allowing `xxx.eps_tex` to include uniformly both `xxx.pdf` and `xxx.eps`. Thus `xxx.pdf_tex` could be deleted and `xxx.eps_tex` moved to `xxx.ptx` which in turn could be included into the main document.

As shown in Figure 4.7, for the current version of `inkscape`, this software filters `xxx.eps_tex` into `xxx.ptx` "manually" so that both `xxx.pdf` and `xxx.eps` are included in `xxx.ptx`. Then it deletes the original files `xxx.pdf_tex` and `xxx.eps_tex`.

The author has filed a bug report to the inkscape-team, to avoid this workaround in future.

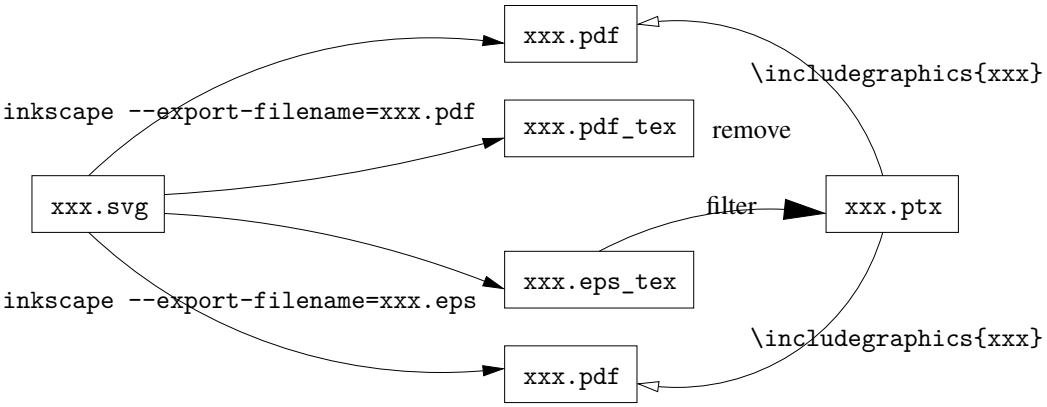
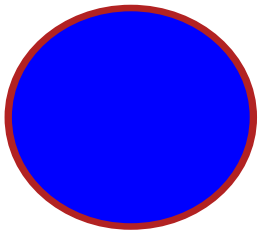


Figure 4.7: Conversion of an svg-file into pdf-, eps- and ptx-files with inclusions



Hello World
From An SVG File!

Figure 4.8: Some svg-picture with text FIXME: uniformity

In contrast to the FIG format, SVG pictures can be created by several programs. Among those is also `inkscape` which can be used like `xfig` as a grapical editor with export functionality. In contrast to FIG format, SVG is essentially human readable, in fact an XML derivative. The author calls it "essentially", referring

to the fact, that the format is quite wordy as is illustrated by the source code `F4_07someSvg.svg` for the above picture. Nevertheless, it can be an advantage to go into internals and manipulate with a text editor. Pasting the into vscode one can see the highlighting, and a preview, of course provided the extensions Section 3.5 and given by Listing 3.2 are installed.

4.6 Pictures which are not transformed

Figure 4.9 shows some picture included as jpg. This is done as usual with the command `\includegraphics` provided by the package `graphicx`. According to the documentation [Car16], page 13, the bounding box must be provided somehow.

This may be done via the package `bmpsize` but alternatively also using the command `ebb`. There is some hint, that `bmpsize` does not work with `xelatex`. So maybe `ebb` is the better alternative. Note that both techniques are available in distribution TEX Live, but not in MiKTeX.

Research shows, that inclusion is seemlessly if PDF files are created. So the problem addressed is specific for creating DVI files. Also at time of this writing, it seems that also in DVI mode, no problems occur. Nevertheless, the author experienced errors on missing bounding box and to be save, provides a way to invoke `ebb` on the file `xxx.jpg`.

With parameter `-m`, this creates a file `xxx.bb` containing the bounding box for `dvipdfm`, and with parameter `-x` a file `xxx.xbb` containing an extended bounding box for `dvipdfmx`. The current implementation seems not to make any difference, whether the bounding boxes are created or not.

Sizes seem to differ in dvi ouput after conversion to pdf, depending on whether `dvipdfm` or `dvipdfmx` is used. Only the latter yields the same size as direct conversion to PDF creates.

Since bounding boxes seem superfluous, we control their creation with a parameter `createBoundingBoxes` whether to invoke `ebb`, which is false by default. Nevertheless, if we invoke, then we do twice creating bounded boxes and extended bounding boxes.

FIXME: further research and further documentation is required.

Note that both for `pdflatex` and siblings creating pdf-output and for `htlatex` in conjunction with `dvipdfmx` (see Section 4.1) files in the format pdf, png, jpg are supported. This list may be incomplete.

As an example, Figure 4.10 shows the same picture as png-file.

FIXME: At the moment, `htlatex` does not work with pictures at all.

Note that in dvi/xdv mode all usual latex converter can include bmp-pictures, whereas in pdf mode only `xelatex` can do that, maybe because it creates XDV internally in any case. In contrast, `lualatex` and `pdflatex` can not.

Figure 4.9: Some jpg-picture, directly included

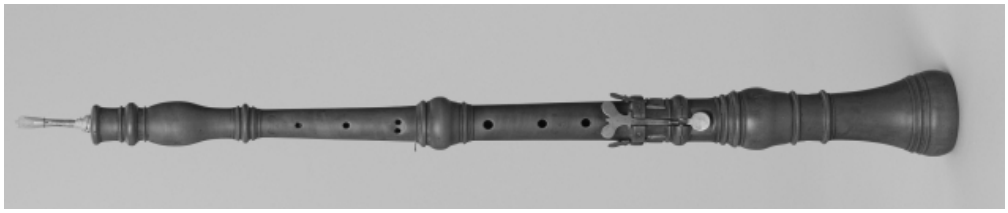


Figure 4.10: Some png-picture, directly included

Chapter 5

Processing of LaTeX Main Files

Given graphics in formats includable in TEX files, which may require preprocessing described in Chapter 4, this section describes the conversions of latex main files into target files in detail. The most important target file format is pdf. Conversion into this format is described in Section 5.1. Note that pdf also occurs as source format for included pictures and as intermediate files. Specific for LaTeX is the dvi format, which is supported mainly for historical reasons.

Almost independent of the format created, Inclusion of bibliographies, indices and glossaries requires additional conversions done by several auxiliary programs. Bibliographies are described in Section 5.2, indices in Section 5.3 and glossaries in Section 5.4. Only at the first sight different but behind the scenes quite analogous is inclusion of results of code evaluations, code in python and other languages described in Section 5.5. Here, an auxiliary program essentially invokes the language interpreter.

Sections 5.6 and 5.7 are special in that they describe rerunning auxiliary programs and the LaTeX-to-pdf converter, which may be necessary even if certain lists are present like table of contents list of figures or list of tables.

Section 5.8 is special in that it is not related with conversion but with checking reproducibility.

Besides the output formats traditional for LaTeX, pdf and dvi describing e.g. books, Section 5.9 describes creation of html, Section 5.10 the creation of odt and Section 5.11 creation of MS Word formats like docx. Finally, also pure text can be generated as described in Section 5.12.

5.1 Transforming LaTeX files into PDF files

The next step is to create a PDF file from the TEX files. LaTeX distinguishes master TEX files from TEX files intended to be inputted from elsewhere. Not taking comments and that like into account, master TEX files roughly have the form

```
\RequirePackage[l2tabu, orthodox]{nag} % optional
\documentclass{...}

\begin{document}
...
\end{document}
```

The core of conversion of a TEX file into a PDF file is running a LaTeX-to-pdf converter `latex2pdf` to a master TEX file `xxx.tex`. The LaTeX-to-pdf converter `latex2pdf` is configurable via the parameter `latex2pdfCommand`. Possible values are `lualatex`, `xelatex` and `pdflatex`, where the first is the default for which this software is also tested. It is also possible to pass parameters to the LaTeX-to-pdf converter. Besides conversion into pdf format, all converters offer conversion to the older dvi format via option `--output-format` as `lualatex` and `pdflatex`, or the alternative extended device independent; an extension of the traditional output format dvi of tex processors, today widely replaced by PDF (dvi) generalizing dvi as `xelatex` does with the option `--no-pdf`.

In fact, the converter `latex2pdf` does much more than converting TEX files to PDF files. Figure 5.1 shows for `latex2pdf` set e.g. to `lualatex`, that besides the PDF file also a LOG file and an AUX file is created. The LOG file contains logging information on the run of the conversion and the aux-file transports information from one run to the next, writing in one run and reading in the next run. Thus, conversion goes without it, but it is read if present. This is why it is depicted at input side in dashed lines.

Optionally, an FLS file is created containing paths to the files the converted LaTeX file depends on and a file with ending `synctex.gz` with information for mapping locations at the created PDF file to the according input files. This is to support backward search, meaning click on a place in the PDF viewer opens an editor in the source file.

What is in fact in the aux-file depends on the package. Among other information, also citations and the location of the bibliography file with ending bib are present. This cannot be used directly in the next `latex2pdf` run to create the bibliography, because the entries referenced in the document must be extracted from the bib-file and sorted. This is done by invoking `bibtex` between two `latex2pdf` runs. Based on the aux-file, `bibtex` creates a bbl-file containing the bibliography, which is read

in the next `latex2pdf` run. For details see Section 5.2.

Alternatively to `bibtex` a bibliography can be created with the package `biblatex` in conjunction with the auxiliary program `biber`. Running a LaTeX-to-pdf converter with package `biblatex` loaded creates a bibliography content file (?): generated by tex processors if used with package `biblatex` (bcf)-file read by `biber`. This software does not support that option. Nevertheless, for sake of completeness we added this data path to Figure 5.1.

If an index is demanded, in addition `latex2pdf` creates a idx-file. As the citations, it cannot be used directly to create an index in the next `latex2pdf` run, because the index entries must be collected and sorted before. This is done by invoking `makeindex` between the two `latex2pdf` runs. Based on the idx-file, `makeindex` creates a index file containing sorted, unified and formatted index entries, output format of `makeindex` and `xindy` (ind)-file containing the index, which is read in the next `latex2pdf` run. For details see Section 5.3.

If more than one index is demanded, we suggest using `splitindex` instead of `makeindex` which creates one ind-file per index.

If a glossary is demanded, this can be read off the auxiliary file: input and output file for tex processors; read also e.g. by `bibtex` (aux)-file and a glossary file containing unsorted and multiple glossary entries; output format of tex processors with package `makeglossaries` (glo)-file containing the index entries is created and a file with style information. Depending on the configuration, this may be a (make-)index style file: output format of tex processors if used with package `glossaries` configured for `makeindex` (ist)-file or a index style file for `xindy`: output format of tex processors if used with package `glossaries` configured for `xindy` (xdy)-file. As for the index the idx-file, the glo-file cannot be used directly to create a glossary in the next `latex2pdf` run, because the glossary entries must be collected and sorted before. This is done by invoking `makeglossaries` between the two `latex2pdf` runs. Based on the glo-file, `makeglossaries` creates a glossary file containing sorted, unified and formatted glossary entries; output format of the `makeglossaries` tool read by tex processors (gls)-file containing the glossary, which is read in the next `latex2pdf` run. For details see Section 5.4.

The package `pythontex` allows including python code or related in the TeX the format, which may also be latex (tex)-file and to evaluate it. The first `latex2pdf` run creates a Code file consisting mainly of code snippets from the tex file; output format of tex processors with package `pythontex` (pytxcode)-file which contains essentially the code parts of the LaTeX-file. Invoking `pythontex` creates by default a folder `pythontex-files-xxx` with material where code is already evaluated. In the next `latex2pdf` run, this material is included in the document. The `pythontex` comes with a second command line utility, `depythontex`, eliminating all python code from the original tex-file. Optionally, `latex2pdf` also creates a File containing

information to replace code snippets in the tex file by the result of their evaluation; output format of tex processors with package `pythontex` if loaded with option `depythontex` (depytxc)-file with all information to replace python code in the original tex-file with evaluated material from `pythontex-files-xxx`. Replacement is done by `depythontex` which by default, sends the result to stdout, but there is an option to write into another LaTeX-file. Converting this new LaTeX-file yields the same result as converting the original one. Depythonization is a feature needed e.g. for papers when the publisher does not accept included code. For details see Section 5.5.

In addition, if a table of contents, a list of figures, a list of tables or a list of listings is required, also a toc-file, a lof-file, a lot-file and a lol-file is created, respectively, collecting the according information. Also, if hyper-references are built, an contains bookmarks: input and output format of tex processors if used with package `hyperref`, file ending seems naive (out)-file containing bookmarks is created. If such a file is present, it is read in and is used to create a table of contents, a list of figures, of tables and of listings or bookmarks in the second run of `latex2pdf`.

To summarize, if a table of contents, a list of figures, a list of tables, a list of listings or a bibliography, an index or a glossary is present, or if code must be replaced by their evaluation, a second LaTeX run is required to make that material appear in the pdf-output.

If a table of contents and at the same time a bibliography, an index or a glossary is present, even two further LaTeX runs are required: After the first one, the bibliography, the index or the glossary occurs in the PDF file but not yet in the table of contents. This happens after the second additional LaTeX run. As described in Sections 5.6 and 5.7, further runs of `makeindex`, resp, `splitindex`, `makeglossaries` and of the LaTeX-processor `latex2pdf` may be necessary.

5.2 Bibliographies

In case the LaTeX-to-pdf converter writes bibliographic information, into its aux-file, a bibliography must be generated. For each occurrence of a `\cite`-command in the TEX file, `latex2pdf` writes an according entry `\citation` into the aux-file. Moreover, a `\bibliography`-command in the TEX file writes a link to the bib-file containing the bibliography data into the aux-file as `\bibdata`. Optionally, a `\bibliographystyle`-command in the TEX file writes a link to the bibliography style file into the aux-file as `\bibstyle`.

To create a bibliography, a `bibtexCommand` must be run after the LaTeX run. The default command is the traditional `bibtex`, but there are more modern alternatives like `bibtexu` and `bibtex8` supporting utf8 encoding and others.

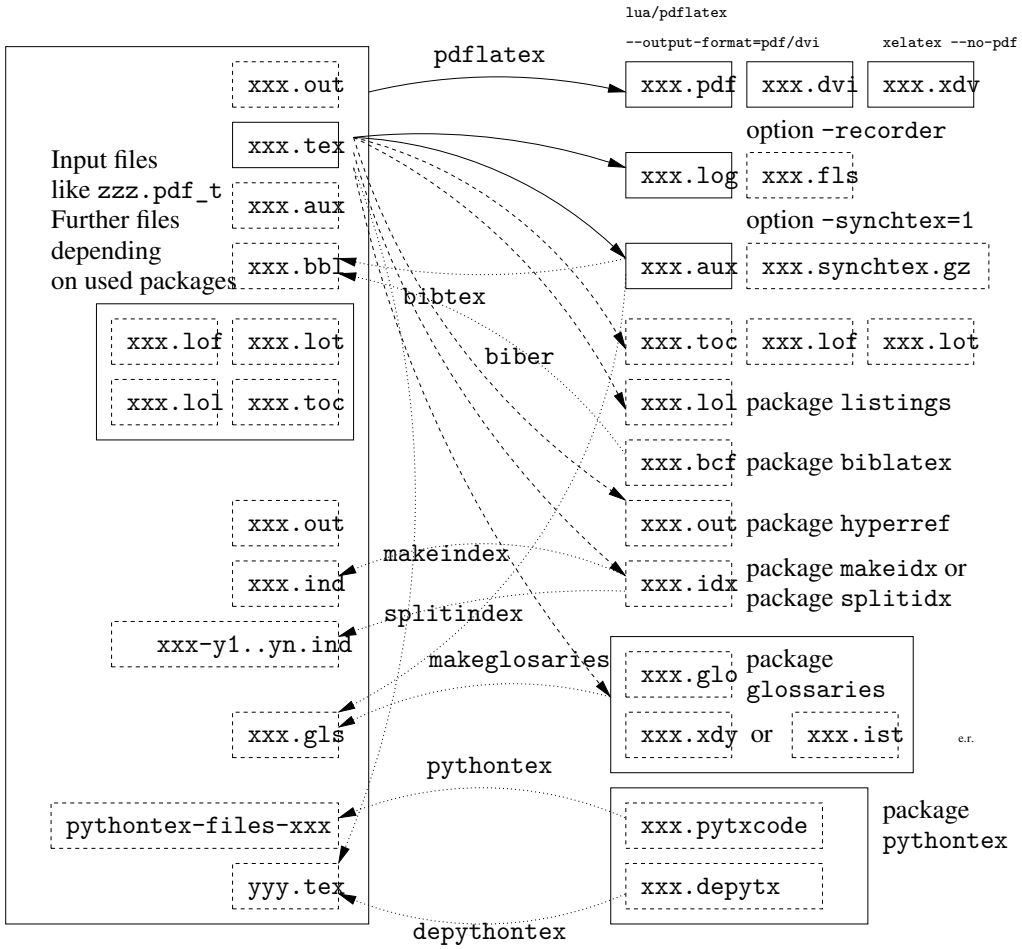


Figure 5.1: Conversion of a TEX file into a PDF, DVI, XDV file

Essentially, `bibtex` extracts the citations in the aux-file, unifies them, i.e. a citation is listed once even if it is used more than once, retrieves the according entries from the bib-file, sorts these, performs formatting and writes all into a bbl-file which can be included in the next run of `latex2pdf`.

Note that after a `bibtex`-run, two LaTeX runs are required: The first one just puts the bibliography found in the bbl-file into the PDF file and the labels of the citations into the aux-file as `\bibcite`-commands. The second run places the labels of the citations found in the aux-file at the citations given by `\cite`. The package `tocbibind` described in [WP10], then writes the headline of the bibliography into the table of contents. The package `rerunfilecheck` described in [Obe16b], ensures that `latex2pdf` is rerun if needed, provided loaded with option `aux`.

This software presupposes, that `bibtex` reads the aux-file and creates a bbl-file and also an blg-file with logging output as illustrated by Figure 5.2. From the

blg-file this software may determine whether `bibtex` emitted an error or warnings.

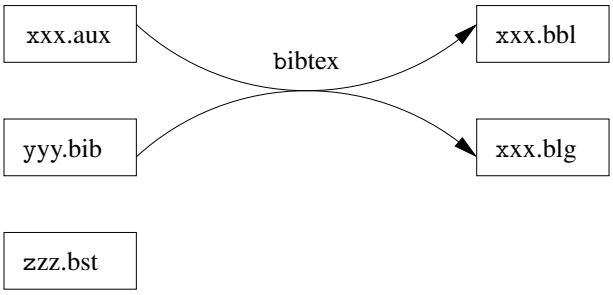


Figure 5.2: Conversion of an aux-file into a bbl-file using a bibliography

Vital information on `bibtex` can be found in [Pat88] and in [Mar09]. Also, Chapter 10 in [Grä96] gives vital information on `bibtex`.

5.3 Indices

In case the LaTeX-to-pdf converter writes index information, into its idx-file, at least one index must be generated. Since the idx-file contains nothing but index information, an index is created if and only if the idx-file is created. Essentially, the command `\makeindex` tells `latex2pdf` to open the idx-file for writing. Then for each occurrence of the `\index`-command or similar (details see below) in the TEX file, an entry is written sequentially to the idx-file as `\indexentry` comprising the keyword given by the `\index`-command and the page number where the `\index`-command occured. For example `\index{ant-task}` creates an entry

```
\indexentry{ant-task|hyperpage}{3}
```

in `xxx.idx`.

Then the `makeindex`-command is applied to the idx-file which sorts keywords and for each keyword collects the according page numbers, sorts it and and writes the result into a ind-file. In the next run of `latex2pdf`, the `\prindindex`-command includes the index as a separate section; typically at the end of the PDF file. The most basic package to provide this command is `makeidx` described in [BLC+14]. In addition, `makeidx` provides the command `\see` which is for cross-reference within an index. The package `tocbibind` described in [WP10], then writes the headline of the index into the table of contents. The package `rerunfilecheck` described in [Obe16b], ensures that `latex2pdf` is rerun if needed, provided loaded with option `aux`.

The same document, [BLC+14] also describes the package `showidx` which prints index entries at the margin of the document. This is for debugging only.

The main restriction of the package `makeidx` is, that only a single index can be created. The reason is that, `latex2pdf` creates a single idx-file and, as illustrated in Figure 5.3, `makeindex` creates a single ind-file from that, representing a single index.

To overcome this restriction, replace package `makeidx` and `makeindex` with package `splitidx` and `splitindex` both described in [Koh16].

The package `splitidx` is used in conjunction with the program `splitindex`. It must be possible to create a single index without using `splitidx` and `splitindex`. ****

Package option `split` makes `latex2pdf` creating idx-files `xxx-y.idx` directly. Here `y` represents the identifier of an individual index. These idx-files can be transformed individually with `makeindex` into ind-files as illustrated in Figure 5.4. Since `latex2pdf` can keep open only up to 16 output streams, not all of which can be used to create a file `xxx-y.idx`, this approach allows a limited number of indices and is thus not recommended and not supported. Another reason is, that this approach undermines the package `rerunfilecheck` described in [Obe16b], and so it is not guaranteed that `latex2pdf` is rerun if needed. This explains why option `split` is not allowed.

Instead, without option `split`, `latex2pdf` creates a single idx-file. The program `splitindex` splits it up into several idx-files and applies `makeindex` to each of them separately as illustrated in Figure 5.5.

For usage of further packages supporting multiple indices which are not intended to be used with this software, see Chapter 8.

This software presupposes, that `makeindex` converts the idx-file into an ind-file containing the index and creating also an ilg-file with logging output as shown in Figure 5.3. From the ilg-file this software may determine whether `makeindex` emitted an error or warnings.

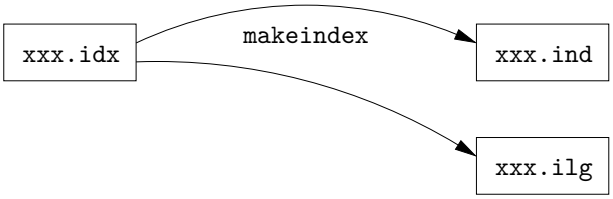


Figure 5.3: Conversion of an idx-file into an ind-file

It is possible to configure the makeindex-command and to pass arbitrary options. CAUTION: For the usual `makeindex`-command, the options `-o` specifying an output file and `-t` (transcript) specifying the logging file are not allowed, because this breaks the expectation to find the sorted index in file `xxx.ind` and bypasses the detection of errors and warnings of this software, respectively. Also specifying a

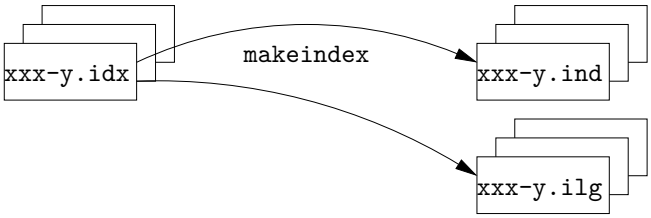


Figure 5.4: Not supported: Conversion of idx-files into ind-files

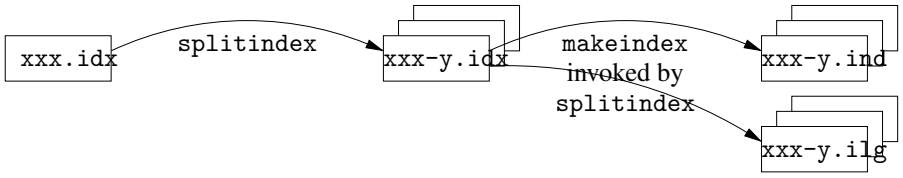


Figure 5.5: Conversion of an idx-file into ind-files

style file via option `-s` is not recommended because this is used to create a glossary and so breaks glossary creation as described in Section 5.4.

Information on the makeindex program can be found in [Mös98] and in [Lam87]. Also, there is a site [LRZ] describing all available options for `makeindex`.

As indicated above, the program `splitindex` invokes `makeindex`. Its options are described in [Koh16], Section 3.10. Since the long option names are not understood in all environments, only the short options are recommended.

Since `splitindex` must satisfy the interface given by Figure 5.5, the option `--help` and its shortcut `-h` are not allowed. Likewise for option `--version` and its shortcut `-V`. The option `--makeindex <makeindex>`, resp. `-m <makeindex>`, is used with the `makeindex` command used for single indices. Thus, this may not be given explicitly but is specified implicitly. Also, the option `--identify <regex>`, resp. `-i <regex>` must be set implicitly because it must be the same expression as used to ***** Then splitindex.tlu is not allowed, because this has another expression.

Only allowable seems `-V`, the shortcut for `--verbose`.

Then comes the name of the index file to be processed without suffix.

The program `splitindex` invokes `makeindex`. The option option `--` coming after the filename, indicates that all following options are passed to `makeindex`

5.4 Glossaries

CAUTION: The method described here, has at least two severe bugs: The number of reruns of the LaTeX-to-pdf converter and also of `makeglossaries` is not guaranteed

as a consequence of a bug in `rerunfilecheck` and the fact, that it does not fit current versions of `makeglossaries`. In addition, entries of the glossaries not mentioned directly in the document but must be included because they are used in the explanation of entries to be included are not treated properly.

As a consequence, this document, or to be more precise its glossary, could not always be reproduced and so the author excluded the glossary until the problem is fixed.

In addition, it is a conceptual weakness that a glossary data base shall be centralized and shall thus not be included in a LaTeX document and not even be written in LaTeX. All weaknesses, bugs and conceptual shortcomings are overcome by the package `glossaries-extra` in conjunction with the auxiliary program `bib2gls` which will replace `glossaries` and `makeglossaries`. For the time being, use glossaries with caution.

Creating glossaries requires the package `glossaries` described in [Tal16]. Note that despite the headline of this section, and despite `glossaries` itself supports multiple glossaries, this software supports only a single glossary and also sorting and unifying is done either via `makeindex` as for indices or via `xindy`, whereas the option to do without external programs offered also by package `glossaries` is not supported by this software.

For generalizations see Chapter 8.

As for creating indices there is a LaTeX-command `\makeindex`, to create a glossary there is a LaTeX-command `\makeglossaries`, but the latter is not built-in as `\makeindex` but provided by the package `glossaries`. If `xxx.tex` is the LaTeX main file, `\makeglossaries` opens the glo-file `xxx.glo` containing glossary entries for writing. As the built-in command `\index` writes entries into the idx-file defining the index, the command `\gls` defined by the package `glossaries` writes an entry into the glo-file. Note that `xxx.glo` typically contains entries more than once and that the entries are not sorted.

To perform sorting, formatting and typically also unification, the package `glossaries` allows three mechanisms. This software supports two of them: via the shell command `makeindex`, which is also used for indices, and via the shell command `xindy`. Using `makeindex` is the default but can also be activated through `\usepackage[makeindex]{glossaries}`. Using `xindy` instead of `makeindex` is triggered through `\usepackage[xindy]{glossaries}`. Accordingly, for option `makeindex` the aux-file receives lines

```
\providecommand\@istfilename[1]{}
\@istfilename{manualLMP.ist}
```

whereas for option `xindy`, there are lines

```
\providecommand\@istfilename[1]{}
```

```
\@istfilename{manualLMP.xdy}
...
\providecommand\@xdylanguage[2]{}
\@xdylanguage{main}{english}
\providecommand\@gls@codepage[2]{}
\@gls@codepage{main}{}
```

This software neither invokes `makeindex` nor `xindy` directly. Instead, it invokes the shell command `makeglossaries` invoked without file ending which determines from the aux-file whether to invoke `makeindex` nor `xindy`. Accordingly, it writes the style definition by creating an ist-file `xxx.ist` or an xdy-file `xxx.xdy` if `makeindex` or `xindy` is specified as package option, respectively.

Seemingly, `makeglossaries` relies on the aux-file to determine whether to invoke `makeindex` or `xindy` for sorting and unification. Then it invokes the according command and writes a LOG file with ending `glg`, redirecting the logging output of `makeindex` or `xindy` adding own output so that a glg-file may be written, even if e.g. `makeindex` is invoked and does not. In any case, if the glg-file is written, `makeglossaries` writes text matching

```
(^\*\*\* unable to execute: )
```

in the glg-file if an error occurs, no matter whether `makeindex` or `xindy` is invoked. Possibly, there are cases where an error causes no glg-file to be written. If no error occurs, a glg-file is written and if warnings are emitted, they either come from `makeindex` or from `xindy`. Thus warnings may be detected with the patterns defined by `makeindex` and by `xindy`.

The style `list` (which is the default) is set in the form

```
\usepackage[style=list]{glossaries}
```

where [Tal16], Section 15 lists predefined styles. So, the style determines the content of the style definition, whereas the options `makeindex` and `xindy` specify the form in which the style is encoded and thus the ending of the style file, which is either `ist` or `xdy`.

Sorting the glo-file, as said above, currently is only supported using the command `makeglossaries`. The allowed options are essentially those making sense for `makeindex` and those making sense for `xindy`. If the shell command `makeglossaries` invokes `makeindex` of course only the according options are passed supplemented by additional options `-s`, `-t`, `-o`, to specify the ist-file, the glg-file (the transcript-file) and the gls-file, respectively, which is the result of sorting, the output file, and contains the entries of the glo-file just sorted, formatted and unified. So for a tex main file `xxx.tex` the program `makeglossaries` invokes

```
makeindex  -s "xxx.ist" -t "xxx.glg" -o "xxx.gls" "xxx.glo"
```

Accordingly, if the shell command `makeglossaries` invokes `xindy` of course only the according options are passed supplemented by additional options `-M`, `-t`, `-o`. This is illustrated in Figure 5.6.

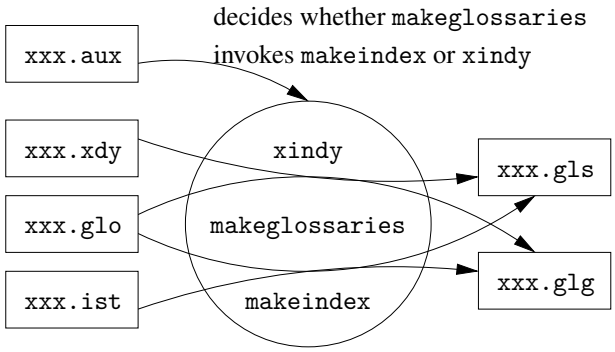


Figure 5.6: Conversion of a glo-file into a gls-file using `makeglossaries`

5.5 Including code via pythontex

The package `pythontex`, described in [Poo21] originally allowed including Python code into a latex document. Later on, further languages were added, most notably octave or Matlab, and the user can easily extended it to further languages as sketched in [Poo21]. Section 7. Of course, to that end, the interpreter for the desired language must be installed. The meaning of the term "including" used above ranges from mere listing to pure execution and comprises also inserting results of execution. A field of application is also creating figures.

Note that like the package `splitindex`, also `pythonindex` comes with an according auxiliary program, in this case, besides `pythontex` also `depythontex`. Consequently, [Poo21] is not only on the package but also on the corresponding command line tools. Since [Poo21] is quite detailed, there is an introduction [Poo] and a gallery [Poo17]. For background on the intentions of package `pythontex`, consult [Poo15]. Information required to integrate pythontex into this software partially goes much beyond the official documentation and is collected in [Rei22]. It could also be interesting for the user for debugging.

Running the LaTeX-to-pdf converter on a file `xxx.tex` with package `pythontex` loaded yields a file `xxx.pytxcode` and if the package is loaded with option `depythontex` also a file `xxx.depytx`. If the file `xxx.pytxcode` is present, this software invokes the command line tool `pythontex` (same name as the according package) to `xxx.pytxcode` (without ending) which converts this into a variety of output files, which are, without further configuration, all in the folder `pythontex-files-xxx` as shown in Figure 5.7, which is described in more detail in [Rei22],

Section 3. Note that this software uses the wrapper `pythontexW` of `pythontex` described below, instead of `pythontex` itself. The figure reflects this.

Running the LaTeX-to-pdf converter again, includes all the output files `*.stdout` in the PDF file or whatever output file created.

An important remark is that `lualatex` is the preferred converter, because files `*.stdout` can impose heavy memory usage and currently `lualatex` is the only converter allocating memory dynamically.

As one can see, `pythontex` cooperates with `lualatex` in a way also `bibtex` or the other auxiliary programs do. Although `pythontex`, at time of this writing in version 0.18, is quite mature, it refrains from writing a log file and indicates errors and warnings just on standard output or error output. This is unlike all the other auxiliary programs in a line with `pythontex`. As a consequence, in particular warnings are difficult to detect and cannot be detected in a uniform way. Thus, the author wrote a little wrapper, called `pythontexW`

```
#! /bin/bash
# bash version >=3.0
pythontex $@ |& tee ${!#}.plg
```

and place it where it can be found, e.g. in the folder of `pythontex`. Note that this is specific for unix-like operating systems but can be easily adapted to windows. Accordingly, `depythontex` behaves in a non-standard way: Firstly, by default, it does not output a result file but outputs on standard output. This can be changed using the option `--output` or `-o` for short. Also, `depythontex` changes into interactive mode if the output file is already present. To avoid this, the option `--overwrite` is required. Overwriting without asking is the standard behavior of all other auxiliary programs. As `pythontex` also `depythontex` does not write a log file but just prints its errors and warnings. Thus, the author wrote a little wrapper, called `depythontexW`

```
#! /bin/bash
# bash version >=3.0
depythontex $@ |& tee ${!#}.dplg
```

and place it where it can be found, e.g. in the folder of `depythontex`.

The package `pythontex` and the according auxiliary programs are highly configurable, more than this software allows.

In particular, in the LaTeX document, the commands `\setpythontexoutputdir` setting the output directory and `\setpythontexworkingdir` setting the working directory shall not be used, because this software assumes the default, that the working directory is the directory containing the LaTeX main file `xxx.tex` and the output directory is in the working directory and its name is `pythontex-files-xxx`.

Further, the package `pythontex` can be configured with package options when loading the package. Since this software is designed for reproducibility, most appropriate would be to specify `runall=true` meaning that even if no python code is modified the auxiliary program `pythontex` executes the python code in the document. Also, it is appropriate to specify `rerun=always`. Note that the defaults are `runall=false` and `rerun=errors`. This behavior makes sense to speed up creation of the document, but it differs from the behavior of all other auxiliary programs and causes the check for update of output files to fail. Moreover, reproducibility is not as easily shown.

The package documentation [Poo21] suggests, that this makes a difference between `runall=true/false` and `rerun=always/errors` if external sources are modified, but as is proved in [Rei22], Section 2.1, the package translates package option `runall=true/false` into key value pair `rerun=always/errors` and this is the only information `pythontex` obtains from the package, so there is no difference.

Also, the auxiliary program `pythontex` itself can be configured via command line arguments. For the package options `runall` and `rerun`, there are according command line options `--runall` and `--rerun` with the same scope. Whereas the package merges options `runall` and `rerun` silently, the auxiliary program `pythontex` emits an error, if both are combined. Essentially one can forget about `runall` and stick to `rerun`.

Strange enough, according to [Poo21], Section 4.1, package options overwrite command line options. This software shall invoke `pythontex` with the option `--rerun=always` which is thus specified as the default. To force unconditional update, this is not sufficient. Instead, this software relies on an undocumented feature of auxiliary program `pythontex` which is likely not to change: If one of the expected output files is missing, it recreates all output files, independent of command line options and package options. Thus, this software deletes one output file if present, before executing `pythontex`.

When this software invokes `pythontex` the exit codes may not be changed via `--error-exit-code`, i.e. if specified then with value `true`. Neither the options `--interactive`, `-h`, `--help` or `--version` are allowed. Currently, this software does not check for options which are not allowed. Fortunately, the latter two command line options have no counterpart in the package configuration.

If we place some code, e.g. python code as inline code using `\pyc`

```
\usepackage[depythontex]{pythontex}
...
\pyc|print(rf'Python inside latex says: "Hello World; 1+1={1+1}"')|
```

the code is really evaluated and the string result is included at proper place as illustrated by the following text which is created by python:

```
Python inside latex says: "Hello World; 1+1=2" .
```

Note that the typewriter font is not created by python, it is explicitly set to highlight the string created by python, but it is python which evaluates the little computation and which prints the string.

Since `pythontex` is written in python, including python code in the LaTeX document uses the python interpreter already installed, as a prerequisite of `pythontex`. To use another language, the according interpreter must be installed in addition to python.

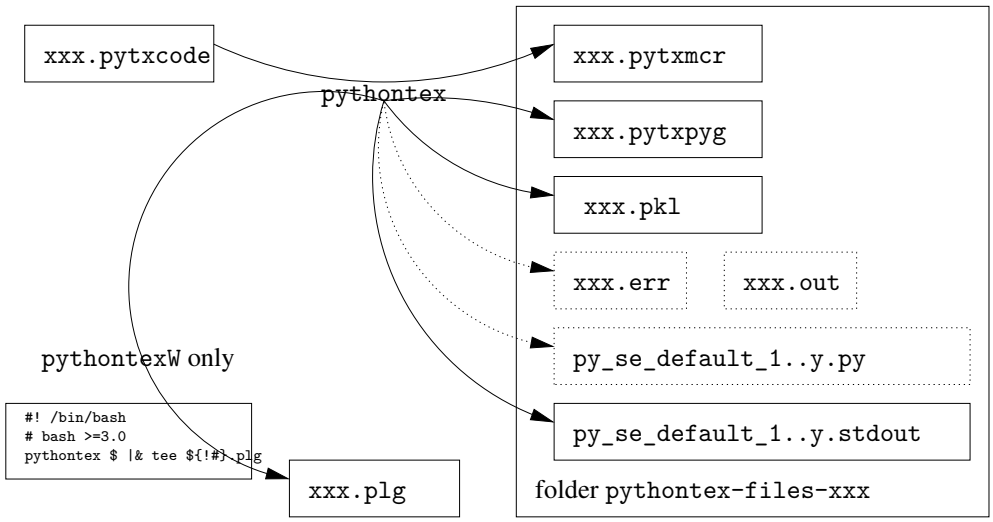


Figure 5.7: Conversion of a `pytxcode`-file using `pythontex`

Figure 5.8 shows the files converted by `depythontex`. As for `depythontex`, this software uses the wrapper `depythontexW` of `depythontex` instead of `depythontex` itself. This is reflected in the figure.

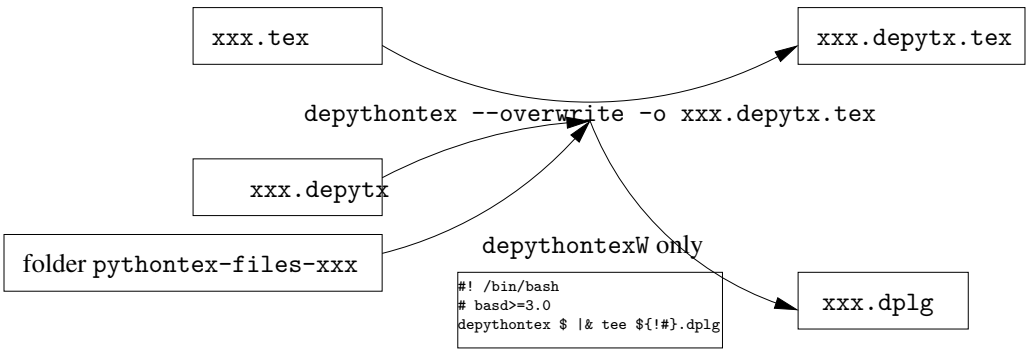


Figure 5.8: Conversion of a `depytx`-file using `depythontex`

5.6 Rerunning the index- and glossary processor

As described in Section 5.1, running a LaTeX-to-pdf converter as `latex2pdf` may detect the presence of a bibliography, an index and/or of a glossary and writes raw files to describe them. After that, an intermediate step is required, sorting, unifying and formatting the entries. This is always done by an external program.

In the next step the LaTeX processor must read in the unified entries again. Whereas a LaTeX-run does not affect neither the bibliography nor python code, it may well invalidate the page numbers of the entries of the index or of the glossary. Thus, the sorted index and glossary must be rebuild before the next LaTeX run makes them visible in the PDF file.

To that end, we use the package `rerunfilecheck`.

```
\usepackage[index, glossary]{rerunfilecheck}
```

must be put before

```
\usepackage{makeidx}
\makeindex
\usepackage[toc, xindy]{glossaries}
%, xindy or even [xindy={language=english, codepage=utf8}]
% mainly for index and glossaries
\makeglossaries
```

in particular before `\makeindex` and `\makeglossaries`.

Note that the package `hyperref` already loads `rerunfilecheck` but with the wrong options. Thus the above declaration must come before

```
\usepackage[...]{hyperref}
```

to avoid error

```
Option clash for package rerunfilecheck
```

Package `rerunfilecheck` detects almost safely changes of the raw index file writing an according message into the LOG file. That way, it can be determined whether it is necessary to rerun `makeindex` and `makeglossaries`. After that, of course LaTeX must be rerun at least once. Note that Section 5.7 describes when to rerun LaTeX without prior running *****

5.7 Rerunning the LaTeX processor

FIXME: a word on change in toc, lof, lot and lol.

As indicated in the previous sections, `latex2pdf` must be rerun, if `bibtex` or `makeindex` `splitindex` or `makeglossaries` had been run to read in the bibliography created by `bibtex` or the index created by `makeindex` or the glossary created by `makeglossaries`. Likewise, if a toc-file, a lof-file, a lot-file or a lol-file had been created in the first `latex2pdf` run, another run is needed to read in these files to create a table of contents, a list of figures or a list of tables, respectively. Note that for all these cases, the LOG file does not allow to detect that `latex2pdf` has to be rerun, by matching a fixed pattern.

After the second run of `latex2pdf`, the table of contents, the list of figures, the list of tables and the list of listings are included and a section with the bibliography, the index and the glossary are inserted. It takes a third run of `latex2pdf` to include the bibliography the index and the glossary into the table of contents. Also it takes that third run to replace the citations with the proper labels given in the bibliography.

Inserting the table of contents, the list of figures, the list of tables and the list of listings may shift the subsequent text which may require another run of `latex2pdf` to get the page numbers right. As described in Section 5.6 intermediate runs of `makeindex` and `makeglossaries` may be required and these also require another run of `latex2pdf` also to get the page numbers right. The package `rerunfilecheck` allows to detect this need to rerun by pattern matching on the LOG file almost for sure: Still there is some chance that the lengths and the md5-sum of all relevant files remain the same, although there is a relevant change. In this case, this software fails to update triggering another `latex2pdf` run.

Note that there are several packages which require additional runs, such as the longtable-package, which may vary dimensions of tables. This software presupposes, that all these reruns may be detected by matching a fixed pattern in the LOG file. Since packages are frequently changed and new packages are written, also the pattern cannot be fixed. Thus it is configurable.

Note that, if a package requires running other programs between two runs of `latex2pdf`, this would require a change in this software.

5.8 Checking reproducibility

There are use cases, where on the one hand side we want to deliver the sources but it is extremely important that the according artifacts are really reproducible. One obvious case is integration test for this software by ensuring that the created artifacts is equivalent with with a confirmed version.

Currently, this is done for pdf files only. The problem with pdf files is, that besides visible contents it contains also meta-data (see [PDF08], Section 14.3), which depends on the run of the conversion. For example the timestamp of conversion

goes into and so do many more aspects.

There are two strategies to deal with the problem:

- make the build process reproducible. The advantage of this is that diffing is quite simple, fast and reproducible: it is byte by byte. This is easily done with a fixed installation but tends to break with update of tools. Also at time of this writing, the different latex engines cannot be treated uniformly.
- use diff tools implementing a weaker notion of equivalence, in a sense visibility equivalence of some degree.

We support both approaches. The first one imposes requirements on the tex source file. It excludes that one displays a creation date changing each run. Listing 5.1 lower part illustrates that: We replaced the creation date by a "version date". A date can be only date of checkin of a version control system or that like. As an alternative, one could also refrain from using dates altogether. In the `\date` command one could display also other pieces of information identifying the document.

But the visible date is not all we have to avoid. Listing 5.1 shows how to eliminate metadata and to replace with adequate one. There is metadata which can be eliminated in a uniform way for all latex engines using the package `hyperref` which the author generally advises. Using `hyperref` with `\hypersetup` one can set metadata such as author, title, subject and keyword as described in [RO22], Section 5.8. On the other hand, one could eliminate `CreationDate` and `ModDate`. For newer versions of `pdflatex` this can be done alternatively with `\pdfinfoomitdate1`.

The metadata which can be eliminated only in a engine specific way are the PTEX keys including the banner, and the `pdftrailerid`. Note that the latter is *intentionally* a unique identifier for the individual document and so strictly speaking it is not advised to eliminate this but eliminating is necessary because it is incompatible with reproducibility. Note that `xelatex` seems not to write PTEX keys at all so only the `pdftrailerid` has to be eliminated or fixed.

Unfortunately, the commands required to make a PDF file reproducible is not uniform among the TEX to PDF converters. In fact, it works best with `lualatex` and sufficiently for `pdflatex` but support seems incomplete with `xelatex`.

If a user renounces independence of the engine and sticks to `pdflatex` only, the package `hyperref` could be replaced by specific commands. Also, using the package `iftex` one can detect the used engine via the macros `\ifLuaTeX`, `\ifXeTeX` and `\ifPDFTeX` as illustrated in Listing 5.2 and adapt the code to the engine.

```
\ifpdf%
  \ifLuaTeX%
    % for lualatex
    \pdfvariable minorversion=7% chktex 1
```

```
% for pdflatex
\pdfsuppressptexinfo-1
% pdfsuppressptexinfo-1 is the same as pdfsuppressptexinfo15
% 1 -> PTEX.Fullbanner
% 2 -> PTEX.FileName
% 4 -> PTEX.PageNumber
% 8 -> PTEX.InfoDict (/Producer /Creator /CreationDate /ModDate /Trapped)
\pdftrailerid{}

% for lualatex:
%   1 -> PTEX.FullBanner
%   2 -> PTEX.FileName
%   4 -> PTEX.PageNumber
%   8 -> PTEX.InfoDict
%  16 -> Creator
%  32 -> CreationDate
%  64 -> ModDate
% 128 -> Producer
% 256 -> Trapped
% 512 -> ID
\pdfvariable suppressoptionalinfo \numexpr32+64+512\relax

% for xelatex
\special{pdf:trailerid [
    <00112233445566778899aabbccddeeff>
    <00112233445566778899aabbccddeeff>
]}

% general
\usepackage{hyperref}
...
\hypersetup{
  pdfinfo={
    Author       ={Ernst Reissner},
    Title        ={The dvi-format and the program dvitype},
    CreationDate={unknown},
    ModDate      ={unknown},
    Subject      ={dvi and dvitype},
    Keywords     ={LaTeX;dvi;dvitype}
  }
}

% alternative for pdftex only
% \pdfinfoomitdate1
% \pdfsuppressptexinfo-1
% \pdftrailerid{}
% \pdfinfo{
%   /Author       (Ernst Reissner)
%   /Title        (The dvi-format and the program dvitype)
%   /CreationDate (unknown)
%   /ModDate      (unknown)
%   /Subject      (dvi and dvitype)
%   /Keywords     (LaTeX;dvi;dvitype)
%}
% Replacing pdfinfoomitdate1 in conjunction with
% usepackage[nodocdata=true,nopdftrailerid=true]{pdfprivacy}
% alternative 1 for pdftex only


\title{The dvi-format and the program dvitype}
\author{Ernst Reissner (rei3ner@arcor.de)}
\date{\versionDate}
```

Listing 5.1: Specifying meta-data for PDF files

```
    % omit CreationDate and ModDate keys.
    \pdfvariable suppressoptionalinfo 767% chktex 1
    % no adding to the trailer dictionary.
    \pdfvariable trailerid{}% chktex 1
    \pdfvariable suppressoptionalinfo -1% chktex 1
  \else
    \ifXeTeX%
      % for xelatex
      \special{pdf:minorversion 7}
      % TBD: find a way to express pdfinfoomitdate: necessary?
      \special{pdf:trailerid []}
    \else
      \ifPDFTeX%
        \pdfminorversion=7          % for pdflatex
        % omit CreationDate and ModDate keys.
        % not before pdfTeX 3.14159265-2.6-1.40.17
        \pdfinfoomitdate=1                    % for pdflatex
        % no adding to the trailer dictionary.
        %\pdftrailer=0                        % for pdflatex
        \pdftrailerid{}                       % for pdflatex
        \pdfsuppressptexinfo=-1               % for pdflatex
      \else
        % Here, the tex processor is unknown.
      \fi%pdftex
    \fi%xetex
  \fi%luatex
```

Listing 5.2: Create reproducible PDF files for various engines

For convenience also `pdfprivacy` described in [Sio17] can be used.

Now that we have described how to ensure reproducible pdf artifacts, just by designing the tex source appropriately, we have to explain how to check that a newly created artifact coincides with a blueprint provided a priori. First of all, note that currently such a check is performed only if the result is in pdf format. Even then the check is performed only if configured so. Then the actual artifacts are compared to predefined artifacts using the configured diff-tool. If the actual artifacts do not coincide with predefined ones according to the chosen diff tool, a build exception is thrown as specified in Table 7.7. The configurations related with diff check are collected in Section 6.12.

5.9 Creating hypertext

To create html and xhtml from TEX files (more precise from LaTeX files), a `tex4htCommand`-command is used Together with its parameters, it is described in Section!6.9. This may be `htlatex`, the default based on `latex` and `htxelatex`

based on `xelatex`.

Figure 5.9 shows the steps `htlatex` performs: From the input LaTeX file `xxx.tex` another LaTeX file `yyy.tex` is created which arises from `xxx.tex` by adding

\usepackage[...]{tex4ht}.

Then `htlatex` runs `latex` on `yyy.tex` which results in `yyy.dvi`. Note that this is in contrast to `lualatex` which would create some `yyy.pdf` unless otherwise specified.

Then comes the converter `tex4ht` into the game which creates several html-files among those also `xxx.html`. The other files, `yyy.idv` and `yyy.lg`, are further processed by `t4ht` creating the stylesheet `xxx.css` and graphic files.

Let us make this more precise. The output of latex is a standard dvi-file interleaved with special instructions for the post-processor `tex4ht` to use. Note that `tex4ht` is the name both of the post-processor and of the LaTeX-package. The special instructions come from implicit and explicit requests made in the source file through commands for TeX4ht.

The utility `tex4ht` translates the dvi-code into standard text, while obeying the requests it gets from the special instructions. The special instructions may request the creation of files, insertion of html code, filtering of pictures, and so forth. In the extreme case that the source code contains no commands of TeX4ht, `tex4ht` gets pure dvi-code and it outputs (almost) plain text with no hypertext elements in it.

The special (`\special`) instructions seeded in the dvi-code are not understood by dvi processors other than those of TeX4ht.

`t4ht` This is an interpreter for executing the requests made in the `xxx.lg` script.

`xxx.idv` This is a dvi-file extracted from `xxx.dvi`, and it contains the pictures needed in the html files.

`xxx.lg` This is a log file listing the pictures of `xxx.idv`, the png files that should be created, CSS information, and user directives introduced through the "`\Needs{…}`" command.

```
(/usr/local/texlive/2014/texmf-dist/tex/generic/tex4ht/tex4ht.4ht
version 2009-01-07-07:11
--------------------------------------
Note --- for additional information, use the command line option `info'
--------------------------------------

(/usr/local/texlive/2014/texmf-dist/tex/generic/tex4ht/html4.4ht

Note: to remove the <?xml version=...?> processing instruction
use the command line option `no-VERSION'

Note: to remove the DOCTYPE declaration
use the command line option `no-DOCTYPE'
)

--------------------------------------
Note: for marking of the base font, use the command line option `fonts+'
Note: for non active _, use the command line option `no_'
Note: for _ of catcode 13, use the command line option `_13'
Note: for non active ^, use the command line option `no^'
```

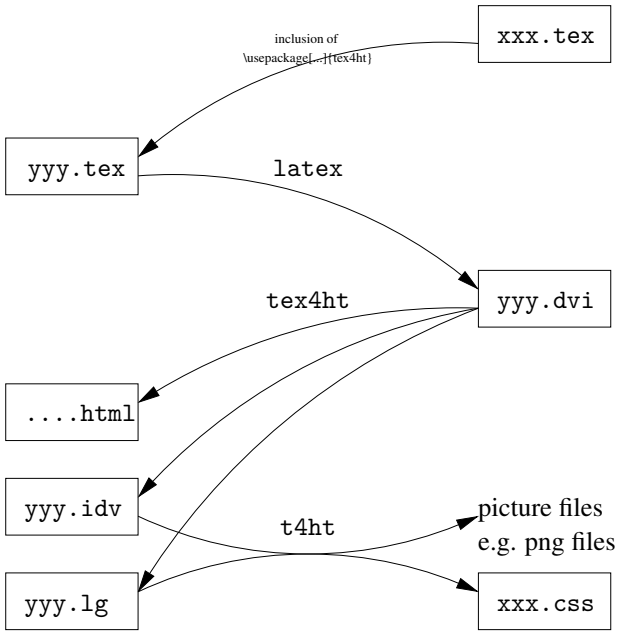


Figure 5.9: Conversion of a TEX file into an xml-file

```
Note: for ^ of catcode 13, use the command line option `^13'
----------------------------------------

(/usr/local/texlive/2014/texmf-dist/tex/generic/tex4ht/html4.4ht
----------------------------------------
Note: For section filenames that reflect on their titles
use the command line option `sec-filename'

Note: for alternative charset, use the command line option `charset=...'

Note: to ignore CSS font decoration, use the `NoFonts' command line option

Note: for jpg bitmaps of pictures,
use the `jpg' command line option.
(Character bitmaps are controled only by `g'
records of tex4ht.env and `-g' switches of tex4ht.c)

Note: for gif bitmaps of pictures, use the `gif' command line option.
(Character bitmaps are controled only by `g'
records of tex4ht.env and `-g' switches of tex4ht.c)

Note: for content and toc in 2 frames,
use the command line option `frames'

Note: for content, toc, and footnotes in 3 frames,
use the command line option `frames-fn'

Note --- for file extension name xht, use the command line option `xht'
----------------------------------------
TeX4ht package options: xhtml,uni-html4,2,pic-tabular,html
----------------------------------------
Note: to ignore CSS code, use the command line option `-css

Note: for inline CSS code, use the command line option `css-in'

Note: for pop ups on mouse over, use the command line option `mouseover'

Note: for addressing images in a subdirectory,
use the command line option `imgdir:.../'
)

Note --- for back links to toc, use the command line option `sections+'
```

```
Note --- for linear crosslinks of pages, use the command line option `next'

(/usr/local/texlive/2014/texmf-dist/tex/generic/tex4ht/latex.4ht
version 2009-05-21-09:32
--------------------------------------
Note --- for links into captions, instead of float heads, use the command l
ine option `refcaption'
--------------------------------------

(/usr/local/texlive/2014/texmf-dist/tex/generic/tex4ht/html4.4ht
--------------------------------------
Note --- For mini tocs immediately aftter the header
use the command line option `minitoc<'

Note --- for enumerated list elements with valued data,
use the command line option `enumerate+'

Note --- for enumerated list elements li's with value attributes, use the c
ommand line option `enumerate-'

Note --- for CSS2 code, use the command line option `css2'

Note --- for bitmap fbox'es, use the command line option `pic-fbox'

Note --- for bitmap framebox'es, use the command line option `pic-framebox'

Note --- for inline footnotes use command line option `fn-in'

Note --- for tracing of latex font commands,
use the command line option `fonts'
--------------------------------------
--------------------------------------
Note --- for width specifications of tabular p entries,
use the `p-width' command line option
or a configuration similar to
\Configure{HColWidth}{\HCode{style="width:\HColWidth"}}
--------------------------------------
)
(/usr/local/texlive/2014/texmf-dist/tex/generic/tex4ht/html4-math.4ht
version 2009-05-18-23:01
--------------------------------------
Note --- for pictorial eqnarray, use the command line option `pic-eqnarray'

Note --- for pictorial array, use the command line option `pic-array'

Note --- for pictorial $...$ environments,
use the command line option `pic-m' (not recommended!!)

Note --- for pictorial $...$ and $$...$$ environments with latex alt,
use the command line option `pic-m+' (not safe!!)

Note --- for pictorial array, use the command line option `pic-array'
)
(/usr/local/texlive/2014/texmf-dist/tex/generic/tex4ht/unicode.4ht
version 2010-12-18-17:40
)
(/usr/local/texlive/2014/texmf-dist/tex/generic/tex4ht/html4-uni.4ht))


(/usr/local/texlive/2014/texmf-dist/tex/generic/tex4ht/html4.4ht
--------------------------------------
Note --- for tocs without * entries, use command line option `notoc*'

Note --- for tocs without * entries, use command line option `notoc*'

Note --- to eliminate mini tables of contents,
use the command line option `nominitoc'

Note --- for frames-like object-based table of contents,
use the command line option `obj-toc'

Note --- for files named derived from section titles,
use the command line option `sec-filename'

Note --- for i-columns index,
use the command line option `index=i' (e.g., index=2)
--------------------------------------
)

(/usr/local/texlive/2014/texmf-dist/tex/generic/tex4ht/html4.4ht

Note --- if included graphics are of degraded quality,
try the command line options `graphics-num' or `graphics-'.
```

```
The `num' should provide the density of pixels in the bitmaps (e.g., 110).

Note --- for key dimensions try the option `Gin-dim';
for key dimensions when bounding box is unavailable
try `Gin-dim+'; neither is recommended
)

(/usr/local/texlive/2014/texmf-dist/tex/generic/tex4ht/html4.4ht
Note --- for URL encoding within href use the command line option `url-enc'
)

(/usr/local/texlive/2014/texmf-dist/tex/generic/tex4ht/html4.4ht

Note --- for pictorial longtable,
use the command line option `pic-longtable'
)

(/usr/local/texlive/2014/texmf-dist/tex/generic/tex4ht/html4.4ht

Note --- to ensure proper alignments use fixed size fonts (see listings.dtx
)
)
```

`tex4ht` yields

```
------------------------------
tex4ht.c (2012-07-25-19:36 kpathsea)
tex4ht
--- error --- improper command line
tex4ht [-f<path-separator-ch>]in-file[.dvi]
   [-.<ext>]            replacement to default file extension name .dvi
   [-c<tag name>]       choose named segment in env file
   [-e<env-file>]
   [-f<path-separator-ch>]        remove path from the file name
   [-F<ch-code>]        replacement for missing font characters; 0--255; default 0
   [-g<bitmap-file-ext>]
   [-h(e|f|F|g|s|v|V)]  trace: e-errors/warnings, f-htf, F-htf search
                            g-groups, s-specials, v-env, V-env search
   [-i<htf-font-dir>]
   [-l<bookkeeping-file>]
   [-P(*|<filter>)]     permission for system calls: *-always, filter
   [-S<image-script>]
   [-s<css-file-ext>]   default: -s4cs; multiple entries allowed
   [-t<tfm-font-dir>]
   [-u10]               base 10 for unicode characters
   [-utf8]              utf-8 encoding for unicode characters
   [-v<idv version>]    replacement for the given dvi version
   [-xs]           ms-dos file names for automatically generated gifs
```

`t4ht` yields

```
-----------------------------------------------------------------------
t4ht [-f<dir char>]filename ...
  -b     ignore -d -m -M for bitmaps
  -c...  choose named segment in env file
  -d...  directory for output files       (default:  current)
  -e...  location of tex4ht.env
  -i     debugging info
  -g     ignore errors in system calls
  -m...  chmod ... of new output files (reused bitmaps excluded)
  -p     don't convert pictures           (default:  convert)
  -r     replace bitmaps of all glyphs    (default:  reuse old ones)
```

```
  -M...  chmod ... of all output files
  -Q     quit, if tex4ht.c had problems
  -S...  permission for system calls: *-always, filter
  -X...  content for field %%3 in X scripts
  -....  content for field %%2 in . scripts

Example:
   t4ht name -d/WWW/temp/ -etex4ht-32.env -m644
------------------------------------------------------------------
```

5.10 Creating odt-files

5.11 Creating MS word files

The best way to convert LaTeX files into MS word files is via ODT files. Conversion from LaTeX to odt is already described in Section 5.10. The last step can be done by `odt2doc` which can create both doc-format and docx-format and many others which is illustrated in Figure 5.10.

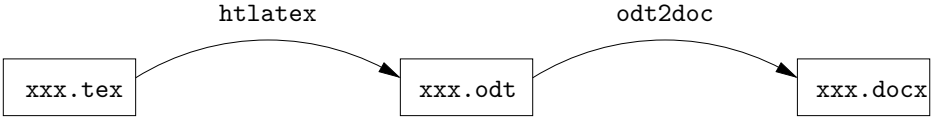


Figure 5.10: Conversion of a TEX file into a docx-file

5.12 Creating plain text files

Why should one create plain text from LaTeX files? Maybe this is the minimal format the receiver can work with. Another common application is word-count, in particular if writing a paper for a journal.

Plain text files can be created from LaTeX files just by stripping off the tex-commands. The disadvantage is, that references, bibliography, index, glossary, table of contents, list of figures, list of tables, …and symbols get lost. Thus, the first step we take is complete creation of a PDF file except display of warnings like bad boxes as described in Section 5.1. This creates an appropriate pdf-file, with correct numbering and links, possibly with overfull boxes and that like. As a final step, we convert the pdf-file into a text file using, as a default `pdftotext` with ending `txt`. Figure 5.11 illustrates the translation process.

Note that `pdftotext` produces a text file with page numbers and signifies the end of a page (to see how, just have a look at the end of the file), so that

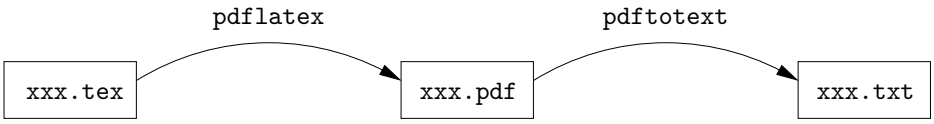


Figure 5.11: Conversion of a TEX file into a txt-file

one can identify page numbers as such. Thus references, index, glossary, table of contents and that like referring to page numbers carry valuable information. Also symbols available in utf8 encoding are preserved. In contrast, heavily stacked formulae become unreadable, because `pdftotext` displays them line by line and drops fraction bars completely. Also, formulae with complex subformulae in a root operator become unreadable because the root operator becomes just a root symbol. Likewise for integrals and that like.

Aspects of figures kept are the captions of course but also the LaTeX-texts. This is displayed line-wise. What gets lost is the postscript/pdf-parts, i.e. the plain graphics.

Chapter 6

Parameters resp. Settings

This section describes the parameters of both the ant-task and the maven-plugin. As this software mainly acts by invoking converters, most parameters refer to commands and options for invocations, but there are general aspects, treated in Section 6.1, but there are also parameters which cannot be associated to an individual converter. These are collected in Section 6.2. All the other sections refer to one or more converters.

The parameters are listed in Tables 6.1 through 6.11 with names, default values and short explanations. Note that neither of the parameters is mandatory, as there are always valid default values.

Each of the tables is described in a separate section. Table 6.1 shows parameter controlling the general conversion process described in detail in Section 6.2. These are directories with names `xxxDirectory` and further parameters not following a naming convention. The other tables show parameters after a certain naming scheme: Command names have the form `xxxCommand` and the parameter with the according options have the form `xxxOptions`. Here `xxx` represents a certain converter. This is one of

fig2dev	The converter of fig-files into mixed latex- and PDF-files.
gnuplot	The converter of gnuplot-files into mixed latex- and PDF-files.
metapost	The converter of MetaPost-files into mixed latex- and PDF-files.
latex2pdf	The converter of latex-files into PDF-files.
bibtex	The creator of a bibliography from an aux-file.
makeindex	The makeindex utility creating an index.
makeglossaries	The makeglossaries utility creating a glossary.

pythontex The utility to invoke python and other languages from within LaTeX and to replace the code by its results dynamically.

depythontex The utility to replace code finally after a run of `pythontex`.

tex4ht The converter of latex into HTML and also into ODT, depending on the parameters.

latex2rtf The converter of latex-files into RTF-files.

odt2doc The converter of ODT-files into doc(x)-files.

pdf2txt The converter of PDF-files into TXT-files.

chktex A code-checker converting in a sense a latex main file into a log-file containing errors, warnings and further messages.

It is a little more complicated with the parameters in Section 6.9.

There are some parameters of the form `patternXxxYyy`, referring to a pattern in the log-file of the converter `Xxx` indicating an event `Yyy` which is one of the following:

ReRun indicates that `Xxx` needs to be rerun.

Err indicates that `Xxx` had an error.

Warn indicates that `Xxx` had a warning.

Essentially, there are two kinds of parameters: Most are just passed to the converters invoked by this software. The parameters of this software are so that the choice of the converter, i.e. the name of the application can be configured, and also each converter can be almost freely configured.

Parameters not passed to an application, are either really crucial or are included to allow also development of latex files.

6.1 Generalities on parameters

As pointed out in the introduction of this chapter, this software acts mainly by invoking various converters. The converters are grouped in so-called *categories*. The converters of a category have the same (file-)interface, means read and write the same files and, mostly but not strictly necessarily, have the same options. For each category there is an option `xxxCommand`, where `xxx` is the name of the

category in lowercase letters[1]. The value of the option is the command to invoke the converter of the category. Also, there is a default converter in each category, and sometimes there is just a single converter possible. For example, `lualatex` is the default converter in the category `latex`.

This software knows about converters and registers the ones approved for this software. Among the advantages are, that it is ensured that the converter is really in that category and that this software checks whether a converter used is in the right category, and it checks whether it is installed in an approved version. On the other hand, there are cases, where the user needs to invoke a custom converter. In this case, the command name shall be given in the form

```
<categoryCommand>commandName:category</categoryCommand>
```

to make sure, that the user is really aware that the converter (s)he uses is in the correct category, i.e. has the required interface. Since neither of the registered converters has a `:` in its name, This form is identified by the occurrence of a colon. Since the categories neither have colons in their names, separation of command name and category is by the last colon occurring. That way, command names may contain colons also.

For most categories of converters, in fact at the time of this writing with a single exception, one can specify command line options, specified in the form

```
<categoryOptions></categoryOptions>
```

In fact, only for `diffPdfCommand` there is no option at all, and for some converters with more complex options, the options are split over more than one setting, e.g. for converter category `fig2dev` converting fig-files, there are general settings given by `fig2devGenOptions` and settings specific for the output language: LaTeX (`fig2devPtxOptions`) and eps (`fig2devPdfEpsOptions`. In any case, options are trimmed, i.e. leading and trailing white spaces are removed before being processed. There are cases, where the options as given are not directly passed to the converter but is further processed. In this case, the processing is documented.

6.2 General parameters and miscellaneous parameters

This section describes the general parameters of Table 6.1 and the miscellaneous in Table 6.2. The latter refer to a single goal only (clear which one).

[1] In fact there are exceptions to this rule: E.g. for category `LaTeX` the command is called `latex2pdf` referring to the common output format pdf, although also dvi and xdv are possible

Parameter Explanation	Default
`texSrcDirectory`	`src/site/tex`
The latex source directory as a string relative to `$baseDirectory`, containing `$texSrcProcDirectory`. This directory determines also the subdirectory of `$outputDirectory` to lay down the generated artifacts. The default value is "`src/site/tex`" on Unix systems.	
`texSrcProcDirectory`	`.`
The latex source processing directory as a string relative to `$texSrcDirectory` containing all tex main documents and the graphic files to be processed and also to be cleaned. Whether this is done recursively in sub-folders is specified by `$readTexSrcProcDirRec`. The default value is "`.`".	
`readTexSrcProcDirRec`	`true`
Whether the tex source directory `$texSrcProcDirectory` shall be read recursively for creation of graphic files, i.e. including the sub-directories recursively. This is set to `false` only during development of documentation.	
`outputDirectory`	`.`
The generated artifacts will be copied to `outputDirectory` relative to `$targetSiteDirectory` which is by default '`$targetDirectory/site`' on Unix systems.	
`targets`	`chk, pdf, html`
A comma separated list of targets to be stored in `$targetSet`. Allowed values are `chk`, `dvi`, `pdf`, `html`, `odt`, `docx`, `rtf` and `txt`. The targets are mostly related to output formats. One exception is `chk` which represents a check, i.e. linting of the source. While in general target `dvi` represents creation of output in DVI format, if `$latex2pdfCommand` is set to `xelatex`, the target `dvi` yields an output in extended DVI format, i.e. in XDV. Also target `html` may represent creation of HTML files and of XHMTL files. Analogously `docx` corresponds with DOCX format by default, but can also be configured to mean DOC. Independent of the order given, the given targets are created in an internal ordering.	
`convertersExcluded`	empty
A comma separated list of excluded Converters given by their command. Excluded converters need not be installed, but their names must be known. They don't show up in the version check of target `vrs` and of course they are not allowed to be used.	
`patternLatexMainFile`	see Section 6.2.1

The pattern to be applied to the contents of tex-files which identifies a LaTeX main file. Here we assume that the latex main file should contain the declaration "`\documentclass`" or the old-fashioned "`\documentstyle`" preceded by a few constructs. Strictly speaking, this is not necessary. For a more thorough discussion, and for an alternative approach, consult the manual.
The default value is chosen to match quite exactly the latex main files, no more no less. Since the pattern is chosen according to documentation collected from the internet, one can never be sure whether the pattern is perfect.
If the current default value is not appropriate, please overwrite it in the configuration and notify the developer of this plugin of the deficiency.

`mainFilesIncluded` empty string

The list of names of latex main files without extension `.tex` separated by whitespace which shall be included for creating targets, except if this is empty in which cases all are included. It is assumed that the names of the latex main files do not contain whitespace. Note that leading and trailing whitespace are trimmed. Currently, names of latex main files should better have pairwise different names, even if in different directories. The empty string is the default, i.e. including all. See parameter `mainFilesExcluded`.

`mainFilesExcluded` empty string

The list of names of latex main files without extension `.tex` separated by whitespace which shall be excluded for creating targets. It is assumed that the names of the latex main files do not contain whitespace. Note that leading and trailing whitespace are trimmed. Currently, names of latex main files should better have pairwise different names, even if in different directories. Together with `mainFilesExcluded`, this is used for document development to build the PDF-files of a subset of documents and e.g. because for a site one needs all documents, but with the software only the manual is shipped. The empty string is the default, i.e. excluding no file. See parameter `mainFilesIncluded`.

`texPath` empty string

Path to the TeX scripts or null. In the latter case, the scripts must be on the system path. Note that in the pom, <`texPath/`> and even <`texPath`> </`texPath`> represent the null-File. The default value is null.

`cleanUp` `true`

Clean up the working directory in the end? May be used for debugging when setting `false`.

`patternCreatedFromLatexMain` see Section 6.2.2

	This pattern is applied to file names and matching shall accept all the files which were created from a latex main file '`xxx.tex`'. It is neither applied to directories nor to '`xxx.tex`' itself. It shall comprise neither graphic files to be processed nor files created from those graphic files. This pattern is applied in the course of processing graphic files to decide which graphic files should be processed (those rejected by this pattern) and to log warnings if there is a risk, that graphic files to be processed are skipped or that processing a latex main file overwrites the result of graphic preprocessing. When clearing the tex source processing directory `$texSrcProcDirectory`, i.e. all generated files should be removed, first those created from latex main files. As an approximation, those are removed which match this pattern. The sequence '`T$T`' is replaced by the prefix '`xxx`'. The sequence '`T$T`' must always be replaced: The symbol '`$`' occurs as end-sign as '`)$`' or as literal symbol as '`$`'. Thus, '`T$T`' is no regular occurrence and must always be replaced with '`xxx`'. Spaces and newlines are removed from that pattern before matching. This pattern may never be ensured to be complete, because any package may create files with names matching its own patterns and so any new package may break completeness. Nevertheless, the default value aims completeness while be tight enough not to match names of files not created. If the current default value is not appropriate, please overwrite it in the configuration and notify the developer of this plugin of the deficiency.

Table 6.1: General parameters

Parameter Explanation	Default
`versionsWarnOnly` Indicates whether the goal `vrs` displays warnings only or also creates pieces of info. Info refers to the version of this plugin and also on its git commit, but also on the versions of the converters found and lists the converters excluded, i.e. those not used and thus not tested on version. Warnings are emitted e.g. if the version of a converter does not fit the expectations, the version of a converter could not be retrieved, e.g. because it is not installed or if the converter specified is unknown altogether. This defaults to `false` displaying also info. The latter is appropriate for using in command line `mvn latex:vrs`, whereas in builds by default the pom overwrites this to have output only in case something goes wrong.	`false`
`injections`	`latexmkrc,chktexrc`

Indicates the files injected by the goal `inj`. This is a comma separated list of `injectionss` without blanks. For further description see Section 3.4.	

Table 6.2: Miscellaneous parameters

6.2.1 The parameter `patternLatexMainFile`

The default pattern for identifying a LaTeX main file is given by Listing 6.1.

```
Strictly speaking, this is not necessary.
For a more thorough discussion,
and for an alternative approach, consult the manual.

The default value is choosen to match quite exactly
the latex main files, no more no less.
Since the pattern is chosen
according to documentation collected from the internet,
```

Listing 6.1: The default pattern of the latex main file in a form as in a pom configuration

This means that the latex main file is the one containing `\documentclass` for modern classes and `\documentstyle` for old-fashioned ones. This may be preceded only by

- the command `\RequirePackage`,
- the command `\PassOptionsToPackage`,
- by comment lines,
- by `\input`, and
- by space.

Note that `\A` represents the beginning of the stream.

Strictly speaking, also `\(re)newcommands` and other constructs are possible but not allowed here. Strictly speaking also `\documentclass` is not required, it could be hidden in some `\input`-file. If we stick to the wise convention to open and close the same environment in the same file and to have `\documentclass`, `\begin{document}` and `\end{document}` in the same file also, then a latex main file without `\documentclass` cannot contain very much material.

So we ask the user to have `\documentclass` declared in the latex main file.

For users of Emacs with package auctex, there is a valuable alternative:

Latex main files are marked with an end section as this file:

```
%%% Local Variables:
%%% mode: latex
%%% TeX-command-extra-options: "-recorder -shell-escape"
%%% TeX-master: t
%%% End:
```

The vital line in this context is `%%% TeX-master: t`. In contrast to this, a non-master file either has no end-section at all or has an end section declaring the according master file (if it is unique) explicitly as the following one from `header.tex`:

```
%%% Local Variables:
%%% mode: latex
%%% TeX-master: "manualLMP"
%%% End:
```

So a pattern for latex main files could also be

```
~%%% Local Variables: $
~%%% TeX-master: t$
~%%% End: ~
\s*
```

For users of visual code/latex workshop, there is another alternative: Absence of first line `% !TEX root`.

Also, if one wants to have a solution not relying on a tool-chain not even on LaTeX tools, One could use a convention, e.g. marking main files with first line `% MAIN FILE`, which makes parsing quite fast.

6.2.2 The parameter `patternCreatedFromLatexMain`

The files created from a latex main file depend strongly on the compiler options and on packages used in the latex main file and in the TEX-files inputted. The default value '`~T$T\.[~.]*`' is appropriate for most parameters and packages: Most packages create files with names only which coincide with the name of the latex main file, except the suffix. This is all sufficient even for programs doing post-processing such as `bibtex`, `makeindex`, `xindy` and `makeglossaries`.

The program `splitindex` requires in addition '`~T$T-.+\.(idx|ind|ilg)`'.

The utility `pythontex` requires '`~T$Tdepytx(\.tex)?`' and creates a bunch of further files all in a folder of the form '`pythontex-files-T$T`' which must also be added to the regular expression.

Whereas typically `latexmk` creates only '`~T$Tfdb_latexmk`' which is included in the very first expression, during its run it creates '`~(pdf|xe|lua)?latex\d+\.fls`', where the digits represent the process number. If interrupting `latexmk`, these files may remain, so it is appropriate to add them to the regular expression.

Package ‘srcltx’ or also synctex requires in addition ‘^T$T\.synctex(\.gz)?’ depending on the setting synctex=1 or synctex=-1. For long files the synctex may create a busy file ‘^T$T\.synctex\(busy\)?’. Even if the LaTeX process is interrupted regularly, at the end the busy file is erased, but still if interrupted from outside it may remain, so we add also the busy variant to the regular expression. Strictly speaking, ‘^T$T\.synctex(\(busy\))?(\.gz)?’ is not precisely what may occur, but is a good approximation.

The class beamer creates a lot of additional files but finally in addition to what we already have, it needs an additional ^T$T\.run\.xml and at times ^T$T\.\d+\.vrb.

Finally, package ‘tex4ht’ is for all the rest of the cases, created by packages.

The pattern is designed to match quite exactly the created files, not much more and at any case not less. In particular, it has to comprise the files matching pattern $patternT4htOutputFiles. Nevertheless, since any new package may break the pattern, and not every package is well documented, this pattern cannot be guaranteed to be final.

If the current default value is not appropriate, please overwrite it in the configuration and notify the developer of this plugin of the deficiency.

The default value for this pattern is currently:

```
^(
T$T                         (\.([^.]*|
                                synctex(\(busy\))?(\.gz)?|
                                out\.ps|run\.xml|\d+\.vrb|
                                depytx(\.tex)?)|
    (-|ch|se|su|ap|li)?\d+\.x?html?|
                        \d+x\.x?bb|
                       \d+x?\.png|
                        -\d+\.svg|
                         -.+\.(idx|ind|ilg))|
      pythontex-files-T$T|
      zzT$T\.e?ps|
      (cmsy)\d+(-c)?-\d+c?\.png|
      (pdf|xe|lua)?latex\d+\.fls)$
```

6.3 Parameters for graphical preprocessing

This section describes the parameters for graphical preprocessing given in Table 6.3.

TODO: do this.

Parameter	Default
Explanation	
`fig2devCommand`	`fig2dev`
The fig2dev command for conversion of fig-files into various formats. Currently, only `PDF` combined with `ptx` is supported.	
`fig2devGenOptions`	empty
The options for the command `$fig2devCommand` common to both output languages. For the options specific for the two output languages '`pdftex`' and '`pdftex_t`', see the explanation of the parameters `$fig2devPtxOptions` and `$fig2devPdfEpsOptions`, respectively.	
`fig2devPtxOptions`	empty
The options for the command `$fig2devCommand` specific for the output language '`pdftex_t`'. Note that in addition to these options, the option '`-L pdftex_t`' specifies the language, `$fig2devGenOptions` specifies the options common for the two output languages '`pdftex`' and '`pdftex_t`' and '`-p xxx.pdf`' specifies the PDF-file to be included.	
`fig2devPdfEpsOptions`	empty
The options for the command `$fig2devCommand` specific for the output language '`pdftex`'. Note that in addition to these option1s, the option '`-L pdftex`' specifies the language and `$fig2devGenOptions` specifies the options common for the two output languages '`pdftex`' and '`pdftex_t`'.	
`gnuplotCommand`	`gnuplot`
The command for conversion of gnuplot-files into various formats. Currently, only `pdf` (graphics) combined with `pdf_t` (latex-texts) is supported.	
`gnuplotOptions`	empty
The options specific for `$gnuplotCommand`'s output terminal "cairolatex", used for mixed latex/pdf-creation. Note that the option '`pdf|eps`' of the terminal '`cairolatex`' is not available, because it is set internally.	
`metapostCommand`	`mpost`
The command for conversion of gnuplot-files into metapost's postscript.	
`metapostOptions`	see Section 6.3.1
The options for the command `$metapostCommand`. Leading and trailing blanks are ignored. A sequence of at least one blank separate the proper options.	
`patternErrMPost`	`(^! )`
The pattern is applied line by line to the log-file of `$metapostCommand` and matching indicates an error emitted by the command `$metapostCommand`. The default value is chosen to match quite exactly the latex errors in the log file, no more no less. Since no official documentation was found, The default pattern may be incomplete. In fact, it presupposes, that `$metapostOptions` does not contain '`-file-line-error-style`'. If the current default value is not appropriate, please overwrite it in the configuration and notify the developer of this plugin of the deficiency.	

`patternWarnMPost`	`^([Ww]arning: )`
The pattern is applied line by line to the log-file of `$metapostCommand` and matching indicates a warning emitted by the command `$metapostCommand`. This pattern may never be ensured to be complete, because any library may indicate a warning with its own pattern any new package may break completeness. Nevertheless, the default value aims completeness while be restrictive enough not to indicate a warning where none was emitted. If the current default value is not appropriate, please overwrite it in the configuration and notify the developer of this plugin of the deficiency. The default value is given below.	
`svg2devCommand`	`inkscape`
The command for conversion of SVG-files into a mixed format.	
`svg2devOptions`	`--export-area-drawing --export-latex`
The options for the command `$svg2devCommand` for exporting SVG-figures into latex compatible files. For more details see Section 6.3.2.	
`createBoundingBoxes`	`false`
Whether for pixel formats like JPG and PNG command `$ebbCommand` is invoked to determine the bounding box. This is relevant, if at all, only in dvi-mode. Note that the package `bmpsize` is an alternative to invoking `ebb`, which seems not to work for `xelatex`. Moreover, all seems to work fine with neither of these techniques. The `$dvi2pdfCommand` given by the default, `dvipdfmx`, seems the only which yields the picture sizes as in PDF mode which fit well. Note also that MiKTeX offers neither package `bmpsize` nor `ebb`. This alone requires to switch off invocation of `ebb` by default. So the default value is `false`.	
`ebbCommand`	`ebb`
The command to create bounding box information from JPG-files and from PNG-files. This is run twice: once with parameter '`-m`' to create '.bb'-files for driver '`dvipdfm`' and once with parameter '`-x`' to create '.xbb'-files for driver '`dvipdfmx`'.	
`ebbOptions`	`-v`
The options for the command `$ebbCommand` except '`-m`' and '`-x`' which are added automatically.	

Table 6.3: Parameters for graphics preprocessing

6.3.1 The parameter `metapostOptions`

The options of the (sole standalone) metapost compiler are given in the metapost manual [Hob14], Appendix B.2.1. The current default option line for this software is as follows:

```
-interaction=nonstopmode -recorder -s prologues=2 -s outputtemplate="%j.mps"
```

The details are as follows:

-interaction=nonstopmode To avoid user interaction in case of an error This seems mandatory.

-recorder Strictly speaking not necessary at the current stage, but for later versions of this software, to allows to track dependencies.

-s prologues=2 In general the `-s` assigns an internal key a value. Here it is the kind of the prologue. The value 2 is a compromise between safe quality of output and length of artifact. As described in detail in [Hob14], Section 8.2, a value of 0 is sufficient for pdf output. Also, if no LaTeX is used to typeset labels, the prologue value is irrelevant. The value 1 is deprecated, 2 yields a prologue only slightly longer than with 0, whereas the safest setting 3 yields a huge prologue. So the compromise is 2 and if 3 is needed in individual cases, this setting can be overwritten in the MP file.

outputtemplate="%j.mps" determines the name of the output file. The default given here uses the jobname and the canonical ending. Unlike the default value of `mpost`, no number of the figure within the metapost file is given. This comes from the fact that we assume a single figure only and ignore the number of the figure.

6.3.2 The parameter `svg2devOptions`

The following options are mandatory:

–**export-area-drawing** Export the drawing (not the page).

–**export-latex** Export into PDF/PS/EPS format without text. Besides the PDF/PS/EPS files, a LaTeX-file `latexfile.tex` is exported, putting the text on top of the PDF/PS/EPS file, i.e. including the according pure graphic file. Include the result in LaTeX as: `\input{latexfile.tex}`.

Note that the latter option is necessary, to create the expected files. It is also conceivable to export text as pdf/eps

The following options are prohibited, because they are automatically added by the software or interfers with:

–**export-filename=FILENAME** Export document to a file with type given by the extension. This is used both to export into PDF and into EPS format. The extension is always given explicitly.

–**export-type=TYPE** Overwrites the type given by `--export-filename`. If no extension is given, this is to determine the export type.

6.4 Parameters for the LaTeX-to-pdf Conversion

This section describes the parameters of the LaTeX-to-pdf converter which are given in Table 6.4.

TODO: do this.

Parameter Explanation	Default
`latex2pdfCommand`	`lualatex`
The LaTeX command to create above all a PDF-file with. Further formats are dvi and xdv and also other formats based on these. Expected values are `lualatex`, `xelatex` and `pdflatex`. Note that for `xelatex` dvi mode (creating xdv-files instead of dvi-files) is not supported, even not creating pdf or other formats via xdv. See also the according options `$latex2pdfOptions` and `$pdfViaDvi`. In particular, this maven plugin does not allow goal `dvi` and related for `xelatex`. Consequently, '`$targets`' may not contain any of these goals.	
`latex2pdfOptions`	see Section 6.4.1
The options for the command `$latex2pdfCommand`. Leading and trailing blanks are ignored. A sequence of at least one blank separate the proper options.	
`patternErrLatex`	`(^! )`
The pattern is applied line-wise to the log-file and matching indicating an error emitted by the command `$latex2pdfCommand`. The default value is chosen to match quite exactly the latex errors in the log file, no more no less. Since no official documentation was found, the default pattern may be incomplete. In fact, it presupposes, that `$latex2pdfOptions` does not contain "`-file-line-error-style`". If the current default value is not appropriate, please overwrite it in the configuration and notify the developer of this plugin of the deficiency.	
`patternWarnLatex`	see Section 6.4.2
The pattern is applied line-wise to the log-file and matching indicates a warning emitted by the command `$latex2pdfCommand`, disregarding warnings on bad boxes provided `$debugWarnings` is set. This pattern may never be ensured to be complete, because any package may indicate a warning with its own pattern any new package may break completeness. Nevertheless, the default value aims completeness while be restrictive enough not to indicate a warning where none was emitted. If the current default value is not appropriate, please overwrite it in the configuration and notify the developer of this plugin of the deficiency.	
`debugBadBoxes`	`true`

Whether debugging of overfull/underfull hboxes/vboxes is on: If so, a bad box occurs in the last LaTeX run, a warning is displayed. For details, set `$cleanUp` to false, rerun LaTeX and have a look at the log-file.

`debugWarnings` `true`

Whether debugging of warnings is on: If so, a warning in the last LaTeX run is displayed. For details, set `$cleanUp` to false, rerun LaTeX and have a look at the log-file.

`pdfViaDvi` `false`

Whether creation of PDF-files from LaTeX-files goes via dvi-files.

If `$pdfViaDvi` is set and the latex processor needs repetitions, these are all done creating dvi and then pdf is created in a final step invoking the command `$dvi2pdfCommand`. If `$pdfViaDvi` is not set, latex is directly converted into pdf.

Currently, not only conversion of LaTeX-files is affected, but also conversion of graphic files into graphic formats which allow inclusion in the tex-file. If it goes via latex, then the formats are more based on (encapsulated) postscript; else on pdf.

In the dvi-file for jpg, png and svg only some space is visible and only in the final step performed by `$dvi2pdfCommand`, the pictures are included using the bounding boxes given by the `.bb` or the `.xbb`-file. These are both created by `$ebbCommand`.

Of course, the target dvi is not affected: This uses always the dvi-format. What is also affected are the tasks creating HTML, ODT or docs: Although these are based on `htlatex` which is always dvi-based, the preprocessing is done in dvi or in pdf. Also the task TXT is affected.

As indicated in `$latex2pdfCommand`, the processor `xelatex` does not create `dvi` but `xdv` files. In a sense, the `xdv` format is an extension of `dvi` but as for he `xdv` format there is no viewer, no way `htlatex` or other applications (except the `xelatex`-internal `xdvidpfmx`) and also no according mime type, we refrained from subsumming this under “kind of dvi”. Thus, with `xelatex` the flag `$pdfViaDvi` may not be set.

`dvi2pdfCommand` `dvipdfmx`

The driver to convert dvi into PDF-files. Note that this must fit the options of the packages ‘`xcolor`’, ‘`graphicx`’ and, provided no autodetection, `hyperref`. Sensible values are ‘`dvipdfm`’, ‘`dvipdfmx`’ and ‘`dvipdft`’, which are all the same in my implementation and ‘`dvipdft`’ (which is roughly a wrapper around ‘`dvipdfm`’ with option `-t` using ‘`gs`’). Note that ‘`dvipdf`’ is just a script around ‘`dvips`’ using ‘`gs`’, but does not provide proper options; so not allowed.

`dvi2pdfOptions` the empty string

The options for the command `$dvi2pdfCommand`. The default value is '`-V1.7`' specifying the PDF version to be created. The default version for PDF format for `$dvi2pdfCommand` is version 1.5. The reason for using version 1.7 is `$fig2dev` which creates PDF figures in version 1.7 and forces `$latex2pdfCommand` in DVI mode to include PDF version 1.7 and finally `$dvi2pdfCommand` to use that also to avoid warnings. Using `$latex2pdfCommand` if used to create PDF directly, by default also PDF version 1.5 is created. For sake of uniformity, it is advisable to create PDF version 1.7 also. In future this will be done uniformly through `\DocumentMetadata` command.
`patternReRunLatex` see Section 6.4.3 The pattern is applied line-wise to the log file and matching triggers rerunning `$latex2pdfCommand` if `$maxNumReRunsLatex` is not yet reached to ensure termination. This pattern may never be ensured to be complete, because any package may indicate the need to rerun `$latex2pdfCommand` with its own pattern and so any new package may break completeness. Nevertheless, the default value aims completeness while be tight enough not to trigger a superfluous rerun. If the current default value is not appropriate, please overwrite it in the configuration and notify the developer of this plugin of the deficiency.
`maxNumRerunsLatex 5` The maximal allowed number of reruns of the LaTeX process. This is to avoid endless repetitions. This shall be non-negative or -1 which signifies that there is no threshold.

Table 6.4: The LaTeX-to-pdf-converter

6.4.1 The parameter `latex2pdfOptions`

An overview over the options supported by the usual latex engines in distribution TeX Live is given in [Rei23], Section 2. In particular, there is a table with the options occurring in any latex converter and columns indicating for each option for which converters it is valid. Note that unlike the other engines, `lualatex` defines options starting with `--`, it works on according options starting with single dash also. To support all engines with the same parameters, the default options are among the ones common to all supported converters. Currently, default option line is as follows:

```
-interaction=nonstopmode -synctex=1 -recorder -shell-escape
```

The details are as follows:

-interaction=nonstopmode To avoid user interaction in case of an error This seems mandatory.

-synctex=1 to create `.synctex.gz` files needed for interaction between editor and viewer.

-recorder Strictly speaking not necessary at the current stage, but for later versions of this software, to allows tracking dependencies.

-shell-escape allows the TEX engine to access the shell to execute. This is needed for some reason for driver `dvipdfmx` which seems to be the sole one supporting PDF-pictures in DVI-mode and PDF-pictures in PDF-mode.

An alternative would be `-shell-restricted`. CAUTION: In MiKTeX this is `--enable-write18` instead.

Note that part of the default values is mandatory, in particular `nonstopmode`, but there are also options which are not allowed. In most of the cases, the problem is that the latex engine does not create an output or does not create it in the expected location or in the expected form. This may apply to the main artifact, i.e. PDF or DVI or XDV, but it may also apply to log files and other files.

The following list of prohibited options is illustrative but not complete:

-draftmode switch on draft mode (generates no output PDF which causes an error)

-output-directory=dir to specify the output directory

-aux-directory=dir to specify the auxiliary output directory

-job-name=name effectively changes the output file name

-quiet makes the log quiet and so circumvents error and warning detection

-fmt=FMTNAME use FMTNAME instead of program name or a %& line

–**luaonly** run a lua file, then exit

-output-format=FORMAT use FORMAT for job output; FORMAT is 'dvi' or 'pdf' pdf is the only allowed …. This is not supported by `xelatex`.

-no-pdf generate XDV (extended DVI) output rather than PDF. This is specific for `xelatex`.

-progname=STRING set program (and fmt) name to STRING only names also without -progname are possible

-help display this help and exit

-version output version information and exit

Note that the default value of `$patternErrLatex` excludes option `-file-line-error-style` and its synonym `--file-line-error-style`. Nevertheless, these options can be used if the pattern `$patternErrLatex` is adapted.

Also option `-halt-on-error` is not strictly forbidden, but not recommended, because it prevents operation as intended for this software.

Two options deserve particular notification, both specifying the output format:

-no-pdf which is specific to `xelatex`, makes `xelatex` create XDV files which currently cannot be further processed by this software. As soon as this software supports XDV files, this option is set by this software, not by the user.

-output-format=FORMAT , which this software uses to set the output format, either to `dvi` or to `pdf`. Strictly speaking, this option is supported by all converters, except for `xelatex`. For `xelatex`, this software only supports `pdf`, which `xelatex` creates because `-no-pdf` is not given. The option `-output-format=pdf` does no harm, because it is ignored. As soon as this software supports XDV creation, it will no longer pass `-output-format` to `xelatex`.

In general, there are two forms of options, one starting with double dash, `--`, and the other form with single dash. In TeX Live, `pdflatex` and `xelatex` use single dash, whereas `lualatex` uses double dash according to the help text. But using the single dash always is ok, because `lualatex` understands single dash also.

In MiKTeX, all options of all converters are double dash. It must be clarified, whether they understand single dash. If not one has to clarify whether in TeX Live all converters understand double dash. If so all must be changed into double dash.

6.4.2 The parameter `patternWarnLatex`

The patterns given below are just by (unwritten) convention. As a consequence, the pattern has a comprehensive default value covering all warnings known to the author, while not detecting a warning, where there is none. To that end, the pattern requires that the warning text starts with the line of the log file. Still the pattern has to be configurable to allow the user to overwrite the default value not being forced to wait for the developer to change it.

For the current default value, we distinguish

- LaTeX-warnings emitted directly by LaTeX starting with `LaTeX Warning:` ,
- LaTeX-font-warnings related with fonts/font selection starting with `LaTeX Font Warning:` ,

- Package warnings emitted by a package. By convention, a package emitting a warning identifies itself by its name `<name>` emitting a warning starting with `Package <name> Warning:` ,
- Class warnings emitted by a package. By convention, a class emitting a warning identifies itself by its name `<name>` emitting a warning starting with `Class <name> Warning:` ,
- pdftex-warning
- Fontspec warnings. Please note the leading character "*".
- Further warnings not identifying themselves as warnings as the word "warning" does not occur.

The resulting default pattern is

```
^(LaTeX Warning: |
LaTeX Font Warning: |
(Package|Class) .+ Warning: |
pdfTeX warning( \((\d|\w)+\))?: |
\* fontspec warning: |
Missing character: There is no .* in font .*!$|
A space is missing\. (No warning)\.)
```

6.4.3 The parameter patternReRunLatex

TODO: add

The pattern to

As a consequence, the pattern has a comprehensive default value covering all warnings known to the author, while not detecting a warning, where there is none. To that end, the pattern requires that the warning text starts with the line of the log file. Still the pattern has to be configurable to allow the user to overwrite the default value not being forced to wait for the developer to change it.

For the current default value, we distinguish

The resulting default pattern is

```
^(LaTeX Warning: Label\(s\) may have changed\. Rerun to get cross-references right\.$|
Package \w+ Warning: .*Rerun( .*|\.)$|
Package \w+ Warning: .*$^\(\w+\) .*Rerun .*$|
LaTeX Warning: Etaremune labels have changed\.$|
\(rerunfilecheck\)               Rerun to get outlines right$)
```

FIXME: There is a bug in this pattern. See Section 9.

6.5 Parameters for creation of the bibliography

This section describes the parameters or creation of the bibliography which are given in Table 6.5.

TODO: do this.

Parameter Explanation	Default
`bibtexCommand`	`bibtex`
The BibTeX command to create a bbl-file from an aux-file and a bib-file (using a bst-style file).	
`bibtexOptions`	empty
The options for the command `$bibtexCommand`.	
`patternErrBibtex`	`error message`
The pattern is applied line-wise to the blg-file and matching indicates that `$bibtexCommand` failed. The default value is chosen according to the '`bibtex`' documentation.	
`patternWarnBibtex`	`^Warning--`
The pattern is applied line-wise to the blg-file and matching indicates a warning `$bibtexCommand` emitted. The default value is chosen according to the '`bibtex`' documentation.	

Table 6.5: The BibTeX-utility

6.6 Parameters for creation of the indices

This section describes the parameters or creation of the indices which are given in Table 6.6.

TODO: do this.

Parameter Explanation	Default
`makeIndexCommand`	`makeindex`
The MakeIndex command to create an ind-file from an idx-file logging on an ilg-file.	
`makeIndexOptions`	the empty string
The options for the MakeIndex command.	
`patternErrMakeIndex`	`(!! Input index error )`
The pattern is applied line-wise to the ilg-file and matching indicates that `$makeIndexCommand` failed. The default value is chosen according to the '`makeindex`' documentation.	
`patternWarnMakeIndex`	`(## Warning )`

The pattern is applied line-wise to the ilg-file and matching indicates a warning `$makeIndexCommand` emitted. The default value is chosen according to the 'makeindex' documentation.
`patternReRunMakeIndex` see Section 6.6.1
The pattern is applied line-wise to the log-file and matching triggers rerunning `$makeIndexCommand` followed by `$latex2pdfCommand`.
This pattern only matches a warning emitted by the package 'rerunfilecheck' e.g. used with option 'index'. The default value is chosen according to the package documentation.
If the current default value is not appropriate, please overwrite it in the configuration and notify the developer of this plugin of the bug.
`splitIndexCommand splitindex`
The SplitIndex command to create ind-files from an idx-file logging on ilg-files. This command invokes `$makeIndexCommand`.
`splitIndexOptions -V`
The options for `$splitIndexCommand`. Here, one has to distinguish between the options processed by `$splitIndexCommand` and those passed to `$makeIndexCommand`. The second category cannot be specified here, it is already given by `$makeIndexOptions`. In the first category is the option '`-m`' to specify the `$makeIndexCommand`. This is used automatically and cannot be specified here. Since `$splitIndexCommand` is used in conjunction with package '`splitidx`', which hardcodes various parameters which are the default values for `$splitIndexCommand` and because the option may not alter certain interfaces, the only option which may be given explicitly is '`-V`', the short cut for '`--verbose`'. Do not use '`--verbose`' either for sake of portability.

Table 6.6: The utilities MakeIndex and SplitIndex

6.6.1 The parameter `patternReRunMakeIndex`

As shown in [Obe16b], Section 2.3, Listing line 166, the pattern is

```
(^\(rerunfilecheck\) +Rerun LaTeX/makeindex to get index right\.$)
```

6.7 Parameters for creation of the Gglossary

This section describes the parameters or creation of the glossary which are given in Table 6.7.

TODO: do this.

Parameter Explanation	Default
`makeGlossariesCommand` The MakeGlossaries command to create a gls-file from a glo-file (invoked without file ending) also taking ist-file or xdy-file into account logging on a glg-file.	`makeglossaries`
`makeGlossariesOptions` The options for the `$makeGlossariesCommand`. These are the options for '`makeindex`' (not for `$makeIndexCommand`) and for '`xindy`' (also hardcoded). The aux-file decides on whether program is executed and consequently which options are used. The default value is the empty option string. Nevertheless, '`xindy`' is invoked as '`xindy -L english -I xindy -M…`'. With option '`-L german`', this is added. Options '`-M<`' for '`xindy`' '`-s`' for '`makeindex`' and '`-t`' and '`-o`' for both, '`xindy`' and '`makeindex`'.	the empty string
`patternErrMakeGlossaries` The pattern is applied line-wise to the 'glg'-file and matching indicates that `$makeGlossariesCommand` failed. The default value '`( unable to execute: )`' is chosen according to the `makeindex` documentation. If the default value is not appropriate, please modify and notify the developer of this plugin.	`(^\*\*\* unable to execute: )`
`patternErrXindy` The pattern in the `makeglossaries` log file (glg)-file which indicates an error when running '`xindy`' via `$makeGlossariesCommand`. If the default value is not appropriate, please modify and notify the developer of this plugin.	`(^ERROR: )`
`patternWarnXindy` The pattern is applied line-wise to the 'glg'-file and matching indicates a warning when running '`xindy`' via `$makeGlossariesCommand`. The default value '`(^WARNING: )`' (note the space and the brackets) is chosen according to the '`xindy`' documentation. If the current default value is not appropriate, please overwrite it in the configuration and notify the developer of this plugin of the deficiency.	`(^WARNING: )`
`patternReRunMakeGlossaries` The pattern is applied line-wise to the log file and matching triggers rerunning `$makeGlossariesCommand` followed by `$latex2pdfCommand`. This pattern only matches a warning emitted by the package '`rerunfilecheck`' e.g. used with option '`glossary`'. The default value is chosen according to the package documentation. If the current default value is not appropriate, please overwrite it in the configuration and notify the developer of this plugin of the bug.	see Section 6.7.1

Table 6.7: The MakeGlossaries-utility

6.7.1 The parameter patternReRunMakeGlossaries

As shown in [Obe16b], Section 2.3, Listing line 210, the pattern is

```
(^\(rerunfilecheck\) +Rerun LaTeX/makeindex to get index right\.$)
```

This holds even if splitindex is used.

6.8 Parameters for including code via pythontex

This section describes the parameters for invoking pythontex and parameters for invoking depythontex which are given in Table 6.8 and in Table 6.9, respectively.

Parameter	Default
Explanation	
pythontexCommand	pythontex
The PythonTeX command which creates a folder pythontex-files-xxx with various files inside from a pytxcode-file (invoked without file ending) and logging in a pythontex log file: home brewed since the original application does not write log files (plg)-file. The default value is pythontex but as long as this does not write a log file this software really needs, we have to configure it with pythontexW which is a simple wrapper of pythontex writing a log file. CAUTION: Since pythontexW is not registered with this software, one has to specify it with its category as pythontexW:pythontex.	
pythontexOptions	--rerun=always
The options for the command $pythontexCommand. For the possibilities see the manual of the pythontex package or the help dialog of pythontex. CAUTION: --rerun and --runall cannot be specified both in one invocation. In the context of this software, the option --interactive is not appropriate. CAUTION: For many options of the command line tool, there is an according package option and the latter overrides the former. CAUTION: This software overwrites settings --rerun and --runall anyway, and forces setting --rerun=always. The default value is --rerun=always.	
patternErrPyTex	see Section 6.8
The pattern in the plg-file indicating that running pythontex, resp. pythontexW via $pythontexCommand failed. The pattern would fit into a single line but because of a bug in pythontex, it is a bit more complicated. If this is not appropriate, please modify and notify the developer of this plugin.	
patternWarnPyTex	see Section 6.8

The pattern in the plg-file indicating a warning when running `pythontex`, resp. `pythontexW` via `$mpythontexCommand`. If this is not appropriate, please modify and notify the developer of this plugin.

Table 6.8: Injecting output of code via `pythontex`

Parameter Explanation	Default
`depythontexCommand` The Depythontex command invoked with no file ending to create a file `xxx.depytx.tex` filefrom a tex-file, a depytxc-file taking the output of `pythontex` into account and logging on a `depythontex` log file: home brewed since the original application does not write log files (dplg)-file. The default value is `depythontex` but as long as this does not write a log file this software really needs, we have to configure it with `depythontexW` which is a simple wrapper of `depythontex` writing a log file. CAUTION: Since `depythontexW` is not registered with this software, one has to specify it with its category as `depythontexW:depythontex`.	`depythontex`
`depythontexOptions`	the empty string

Table 6.9: Replacing code by its output via `depythontex`

The pattern `patternErrPyTex` is by default

```
\*␣PythonTeX␣error|...
```

substituting the dots by

```
(PythonTeX:␣␣.+␣-|␣␣␣␣-␣Current:␣)␣[1-9][0-9]*␣error\(s\),␣[0-9]+␣warning\(s\)
```

Accordingly, the pattern textttpatternWarnPyTex is by default

```
(PythonTeX:␣␣.+␣-|␣␣␣␣-␣Current:␣)␣[0-9]+␣error\\(s\),␣[1-9][0-9]*␣warning\(s\)
```

6.9 Parameters for conversion LaTeX to HTML

This section describes the parameters of the LaTeX-to-html converter which are given in Table 6.10.

Parameter Explanation	Default
`tex4htCommand`	`htlatex`
`tex4htStyOptions`	`xhtml,uni-html4,2,svg,pic-tabular`
`tex4htOptions`	`' -cunihtf -utf8'`
`t4htOptions` The options for '`t4ht`' which converts idv-file and lg-file into css-files, tmp-file and, by need and if configured accordingly into png-files. The value '`-p`' prevents creation of png-pictures.	the empty string
`patternT4htOutputFiles` The pattern is applied to file names and matching shall accept exactly the target files of goal '`html`' for a given latex main file '`xxx.tex`'. Matching triggers copying those files to `$outputDirectory`. The patterns for the other targets are hardcoded and take the form '`^T$T\.yyy$`', where '`yyy`' may be an ending or an alternative of endings. This pattern is neither applied to directories nor to '`xxx.tex`' itself. For an explanation of the pattern '`T$T`' see `$patternCreatedFromLatexMain`. Spaces and newlines are removed from that pattern before processing. The pattern is designed to match quite exactly the files to be copied to `$targetSiteDirectory`, for the goal '`html`', not much more and at any case not less. Since `$tex2htCommand` is not well documented, and still subject to development, this pattern cannot be guaranteed to be final. If the current default value is not appropriate, please overwrite it in the configuration and notify the developer of this plugin of the bug.	see Section 6.9.1

Table 6.10: The LaTeX-to-html-converter

6.9.1 The parameter `patternT4htOutputFiles`

The default value has the following components:

- '`^T$T\.x?html?$`' is the main output file.
- '`^T$Tli\d+\.x?html?$`' are lists: toc, lof, lot, indices, glossaries, NOT the bibliography.
- '`^T$T(ch|se|su|ap)\d+\.x?html?$`' are chapters, sections and subsections or below and appendices.

- '`^T$T\d+\.x?html?$`' are footnotes.
- '`^T$T\.css$`' are cascaded stylesheets.
- '`^T$T-\d+\.svg$`' and '`^T$T\d+x\.png$`' are svg/png-files representing figures.
- '`^T$T\d+x\.x?bb`' are the bounding boxes (suffix `.bb` for `dvipdfm` and suffix `.xbb` for `dvipdfmx`).
- '`^(cmsy)\d+(-c)?-\d+c?\.png$`' represents special symbols.

Note that the patterns for the html-files can be summarized as

```
^T$T((ch|se|su|ap|li)?\d+)?\.x?html?\$
```

This altogether constitutes the default value for this pattern:

```
^(T$T(((ch|se|su|ap|li)?\d+)?\.x?html?|
\.css|
\d+x\.x?bb|
\d+x\.png|
-\d+\.svg)|
(cmsy)\d+(-c)?-\d+c?\.png)$
```

The pattern is designed to match quite exactly the files to be copied to `$targetSiteDirectory`, for the goal "html", not much more and at any case not less. since `$tex2htCommand` is not well documented, and still subject to development, this pattern cannot be guaranteed to be final.

6.10 Parameters for further conversions

This section describes the parameters of the converter from and to further formats which are given in Table 6.11.

These converters convert latex into RTF directly, they convert ODT into doc-like documents and pdf into pure text. A special case is the code-checker in a sense converting latex into a log-file. For each of them, the name of the command can be specified and also the options. Since neither of them, except the code checker, write a log-file, there are no further parameters necessary.

Parameter Explanation	Default
`latex2rtfCommand`	`latex2rtf`
The latex2rtf command to create RTF from latex directly.	
`latex2rtfOptions`	the empty string
The options of the command `$latex2rtfCommand`.	
`odt2docCommand`	`odt2doc`
The odt2doc command to create MS word-formats from otd-files.	
`odt2docOptions`	`-fdocx`
The options of the command `$odt2docCommand`. Above all specification of output format via the option '-f'. Invocation is '`odt2doc -f<format> <file>.odt`'. All output formats are shown by '`odt2doc --show`' but the formats interesting in this context are the following: `doc`, `doc6`, `doc95`, `docbook`, `docx`, `docx7`, `ooxml` and `rtf`. Interesting also the verbosity options '`-v`', '`-vv`', '`-vvv`' the timeout '`-T=secs`' and '`--preserve`' to keep permissions and timestamp of the original document.	
`pdf2txtCommand`	`pdftotext`
The pdf2txt-command for converting PDF-files into plain text files.	
`pdf2txtOptions`	the empty string
The options of the command `$pdf2txtCommand`.	

Table 6.11: The parameters of further converters

FIXME: Note that `pdftotext -h` prints a usage message. This is a way to obtain not the specified output. It shows that `pdftotext -q` does not print any messages or errors. This indicates that `pdftotext` normally does display error messages on the standard output. These may be led to a log file to indicate errors and warnings. Here, further research is required.

The option `-htmlmeta` seems not approprate. The option resolution `-r` seems sensible only in conjunction with the crop area defined by `-x` and `-y` which does not make sense in our context. The same holds for specification of the first and the last page via `-f` and `-l`. What does make sense is specifying the encoding via `-enc` with possible values given by `pdftotext -listenc`. What makes sense most is `UTF-8`.

6.11 The code checker `chktex`

Among the applications used by this software, the codechecker plays a special role: it is not really a converter, unless we interprete the log file as artifact. Like for the most converters also for the codechecker we can specify the command ant its

options.

Parameter Explanation	Default
`chkTexCommand` The chktex-command for checking latex main files.	`chktex`
`chkTexOptions` The options of the command `$chkTexCommand`, except "`-o output-file`" specifying the output file which is added automatically. For further details see the options below.	`-q -b0`

Table 6.12: The parameters of the code checker

The options of `chktex` are described in detail in [Thi22], Section 6.1.2.

Here is a list of options useful in this context. The first group of these are muting options:

- '-w', '-e', '-m', Make the message number passed as parameter a warning/an error/a message and turns it on. Messages are not counted.
- '-n' Turns the warning/error number passed as a parameter off.
- '-L' Turns off suppression of messages on a per line basis.

The next group of interesting options are for output control:

'-q' Shuts up about copyright information.

'-o output-file' Specifies the output file. This is added automatically and shall thus not be specified by the user.

'-b0/1' If you use the -o switch, and the named output-file exists, it will be renamed to 'filename.bak' for option `-b1` and not for `-b0`.

'-f format' Specifies the format of the output via a format similar to "`printf()`'. For details consult the manual [Thi22], Section 6.1.2. The codes are listed below.

'-vd' Verbosity level followed by a number 'd' specifying the format of the output according to the listing below. The verbosity number is resolved as a pattern as if given by the option '-f format'. Thus the option '-v' is ignored if the option '-f format' is specified.

The default value `-q -b0` avoids verbose output and backing up the output log-file.

Code

%b String to print between fields (from -s option).

%c Column position of error.

%d Length of error (digit).

%f Current file-name.

%i Turn on inverse printing mode.

%I Turn off inverse printing mode.

%k kind of error (warning, error, message).

%l line number of error.

%m Warning message.

%n Warning number.

%u An underlining line (like the one which appears when using '-v1').

%r Part of line in front of error ('S'-1).

%s Part of line which contains error (string).

%t Part of line after error ('S'+1).

FIXME: to be inserted. See [Thi22], Section 6.1.6. From `chktexrc`:

```
OutFormat
{
# -v0; silent mode
%f%b%l%b%c%b%n%b%m!n

# -v1; normal mode
"%k %n in %f line %l: %m!n%r%s%t!n%u!n"

# -v2; fancy mode
"%k %n in %f line %l: %m!n%r%i%s%I%t!n!n"

# -v3; lacheck mode
"!"%f!", line %l: %m!n"

# -v4; verbose lacheck mode
"!"%f!", line %l: %m!n%r%s%t!n%u!n"
```

```
# -v5; no line number, ease auto-test
"%k %n in %f: %m!n%r%s%t!n%u!n"

# -v6; emacs compilation mode
"!"%f!", line %l.%c:(#%n) %m!n"
}
```

Note that "`!`" is to escape quotes and newline. More than these can be added to `chktexrc`.

This document is checked with options deviating from the default value:

```
-q -b0 -v1 -g0 -l ${basedir}/src/site/tex/chktexrc
```

The default is `-q -b0`, option `-g0` means that the global `chktexrc` is not used and option

```
-l ${basedir}/src/site/tex/chktexrc
```

specifies a record file tailored to the needs of this project. In particular, the pattern for `-v1` is slightly modified: It is

```
# -v1; normal mode
"%k %n in %f line %l: %m!n %r%s%t!n %u!n"
```

which adds a blank to all lines but the headlines. That way, the kind of issue (%k) is easily parsed. This could be used for emitting an error instead of a warning when processing goal *check*.

Although the return code of `chktex` is not documented, a bit of reverse engineering shows the following distinction:

0. Successful execution and found neither an error nor a warning.
1. Execution as such did not succeed, e.g. because of an invalid option like `-exx`.
2. An error occurred and in particular execution as such suceeded.
3. A warning occurred but no error and in particular execution as such suceeded.

On this behavior this software bases its failure messages.

6.12 Parameters for ensuring reproducibility

For a general description of the reproducibility check see Section 5.8. Here we go into the details and identify the parameters controlling the check and specified in great detail in Table 6.13. As already mentioned in Section 5.8, currently, checks are performed for artifacts in pdf format only; more formally, if the target (which is in parameter `target` described in Table 6.1) is `pdf`.

But if so, the parameter `chkDiff` decides whether a check is performed at all. Note that checking is off by default. Then a diffing tool given by `diffPdfCommand` expects the blueprints in the directory `diffDirectory`. In contrast, the actual artifacts to be checked are in `outputDirectory`, whereas the sources are in `texSrcDirectory`.

The location of a source tex file relative to `texSrcDirectory` is the location of the artifact relative to `outputDirectory`. This path relative to `diffDirectory` is the location of the blueprint. With the actual artifact in `outputDirectory` and the blueprint in `diffDirectory` the diff-tool determines whether the both are equivalent. If so, equivalence is logged as an info, else an exception described in Table reftab:TLP is thrown.

Note that the choiced of the diff tool `diffPdfCommand` determines teh notion of equivalence of the pdf artifacts, ranging from byte equivalence to some kind of visual equivalence.

Parameter	Default
Explanation	
`diffDirectory`	.
Diff directory relative to `$baseDirectory` used for diffing actually created artifact against prescribed one inthis directory. This is relevant only if `$chkDiff` is set. The default value is ..	
`chkDiff`	`false`
Indicates whether after creating artifacts and copying them to the output directory `$outputDirectory` the artifacts are checked by diffing them against preexisting artifacts in `$diffDirectory` using the diff command given by `$diffPdfCommand`. Note that currently, only pdf files are checkd. This is `false` by default and is set to `true` only in the context of tests.	
`diffPdfCommand`	`diff`

The diff-command for diffing PDF-files strictly or just visually to check that the created pdf files are equivalent with prescribed ones. CAUTION: There are two philsophies: Either the latex source files are created in a way that they reproduce strictly. Then a strict diff command like `diff` is appropriate. Else another diff command is required which checks for a kind of visual equality. The default value is a mere `diff`. Alternatives are `diff-pdf` and `diff-pdf-visually` both implementing a visual diff. Note that unlike for other tools, no options can be passed in this case explicitly.

Table 6.13: The parameters of the pdf differ

The options of `chktex` are described in detail in [Thi22], Section 6.1.2.

Chapter 7

Exceptions and Logging

If during execution of this software something goes wrong, and it is possible to detect that, the user shall be notified.

Maven foresees a mechanism to abort the whole build, i.e. lifecycle phase or a single goal and accordingly ant allows to abort a task. In both cases, abortion is implemented by throwing an exception.

A maven plugin aborts a goal throwing a

`org.apache.maven.plugin.MojoFailureException`

and a

`org.apache.maven.plugin.MojoExecutionException`

to abort the life-cycle phase. Since this plugin is just for documentation, there is no need to abort site creation altogether, so only the former exception occurs.

An ant-task aborts an ant-build throwing a

`org.apache.tools.ant.BuildException`

without further distinction.

This software provides both a maven plugin and an ant task built on the same code base. Thus, the maven plugin throws a `MojoFailureException` if and only if the according ant-task throws an `BuildException` in the same situation.

Section 7.1 describes the philosophy of throwing an exception and defines in detail under what circumstances which exception is thrown.

Roughly speaking, an exception is thrown only if something is really wrong, e.g. a non-recoverable error or an indication that the build system is out of control or if this plugin/task is likely to destroy the work of another plugin/task.

If something went wrong, but no exception is thrown, the user must be notified by logging and the build process to go on, skipping a section of a task as small as

possible. Both, maven and ant provide a logging mechanism with the levels error, warning, info and debug. Section 7.2 describes the errors and warnings; the lot of infos and debugging output are not described here.

Verbosity is chosen by the following command line options:

`-e` shows error messages,

`-X` shows debug-messages,

`-q` quiet hides the info-level and shows *only* errors.

There seems no way to get warnings only.

Each exception offers a message and also each warning has a warning message. The messages are endowed with a unique identifier of the form `KCCDD`, where `K` is the kind which is one of

T Throwable, which results in a `MojoFailureException` for the maven-plugin and `BuildException` for the ant-task. This is described in detail in Section 7.1

E logging as ERROR,

W logging as WARNING

I logging as INFO which occurs frequently

D logging as DEBUGGING output, which is lengthy

The shortcut `CC` describes the class where the exception is thrown or the warning is logged:

EX CommandExectutorImpl: a class executing applications on a command line.

PP LatexPreProcessor: preprocessing of LaTeX-files: Processing of graphic files and detection of latex main files.

LP LatexProcessor: processing of LaTeX-main files: conversion into various output formats.

SS Settings: A container holding the values of all parameters. These are either default or read from the configuration in the pom for the maven plugin and in the build file for the ant task.

MI MetaInfo: offering meta information as expected and actual versions of converters.

FU TexFileUtilsImpl: a class providing access to files.

Finally, `DD` is a two digit number enumerating the messages.

Identifier	Message Explanation
`WMI01`	`Version string from converter $conv did not match expected form: $conv: 'version'not?in$interv` Indicates that the version string coming from the converter $conv is not as expected. Programming error excluded, this means that the version does not fit, i.e. is not in `$interv`.
`WMI02`	`$conv: '$actVersion'not in$interv` Indicates that the version of converter $conv can be detected and is $actVersion but does not fit the expectation which is $expVersion.

Table 7.1: The logging for MetaInfo

Identifier	Message Explanation
`WFU01`	`Cannot read directory '$dir'; build may be incomplete.` TBD
`XFU02`	`TBD` TBD
`WFU03`	`Cannot close '$file'.` TBD
`EFU05`	`Cannot delete file '$file'.` TBD
`EFU06`	`Cannot move file '$src' to '$dest'.` TBD
`EFU07`	`File '$srcFile' to be filtered cannot be read.` WORKAROUND for `inkscape` filtering `eps_tex`-file into `ptx` file: The former is not a readable regular file.
`EFU08`	`Destination file '$destFile' for filtering cannot be written.` WORKAROUND for `inkscape` filtering `eps_tex`-file into `ptx` file: The latter is not a writable regular file.
`EFU09`	`Cannot filter file '$srcFile' into '$destFile'.` WORKAROUND for `inkscape` filtering `eps_tex`-file into `ptx` file: Either reading a line or writing a line failed.
`WFU10`	`Cannot overwrite/clean file '$aFile' because it is not self-created.`

	May occur if a file, e.g. `.latexmkrc` is present in the latex source directory and is not created by this software. To avoid the risk of overwriting or deleting user-written files, only config files written by this software can be overwritten in goal `inj` or deleted in goal `clr`.
`WFU11`	`Refuse to overwrite/clean file '$aFile' because it may be not self-created or has dangling reader.` To avoid the risk of overwriting or deleting user-written files, this software checks whether it was this software which wrote the files by reading the headline. If this is not possible or if the reader to read that headline could not be closed after reading, this warning is emitted. Neither is the file overwritten in goal `inj` nor is it deleted in goal `clr`.

Table 7.2: The logging for TexFileUtils

TBD: check whether workaround still necessary. TBD: complete list TBD: add missing lists

7.1 Exceptions

Exceptions are thrown only if no substantial part of this maven-goal or this ant-task may be completed as if the tex source directory does not exist or is no directory or if a failure occurs which indicates that the underlying system does not work properly, as if the tex source directory or a subdirectory is not readable or if execution of an external program fails. The latter does not mean that the program returns with an error code, but it means that execution from within java fails.

Identifier	Message Explanation
`TEX01`	`Error running $command.`

Compare with EEX01 in Table 7.9: Error execution means

- the file expected to be the working directory does not exist or is not a directory.
- method `Runtime.exec(String, String[], File)` fails throwing an `IOException`.
- an error inside systemOut parser occurs
- an error inside systemErr parser occurs
- Wrapping an `InterruptedException` on the process to be executed thrown by `Proces.waitFor()`.

whereas for EEX01 just a failure code is returned.

Table 7.3: The `BuildFailureException`s of the class `CommandExecutorImpl`

Identifier	Message Explanation

`TSS01`	`The tex source directory '$texSrcDirectoryFile'`
	`should be an existing directory, but is not.`
	The tex source directory is given in the pom/build-file with default value `./src/site/tex`. It contains or is `$texSrcProcDirectoryFile`. Thus is must be a directory.

`TSS02`	`The tex source processing directory '$texSrcProcDirectoryFile'`
	`should be an existing directory, but is not.`
	The tex source processing directory is given in the pom/build-file relative to `$texSrcDirectoryFile` with default value `.`. It contains all files to be processed. Thus is must be a directory.

`TSS03`	`The output directory '$outputDirectory'`
	`should be a directory if it exists, but is not.`
	The output directory is given in the pom/build-file with default value `./target/site/.`. The output directory is where the result of the goal/-task are copied to. If it does not yet exist, it is created but if it exists and is a regular file, it cannot be created any more.

`TSS04`	`The target set '$targets'` `should be a subset of the registered targets '…', but is not.`

The target set is given in the pom/build-file with default value `pdf, html`. A single target can be given on the command line as e.g. via `mvn latex:pdf` and also in this case, the validity of the target is checked, so that e.g. `mvn latex:invalid` throws an exception, but the mechanism relies directly on mavens ability to check the targets of this plugin.
`TSS05` `The excluded converters '$convertersExcluded' should form a subset of the registered converters '…'.`
From the possible "registered" converters the ones not used may be excluded to avoid that they cause errors when trying to check correctness of version in target `vrs` accessed via `mvn latex:vrs`. These converters may not even be installed.
`TSS06` `Tried to use converter '$convStr' although not among the registered converters '…' as expected.`
Only registered converters may be used.
`TSS07` `Tried to use converter '$convStr' although among the excluded converters '…'.`
Among the registered converters only those may be used, which are not excluded, i.e. listed in configuration in section `convertersExcluded`.
`TSS08` `Tried to use converter '$convStr' in configuration '…' instead of configuration '…'.`
Each converter may occur in a specified configuration only. So e.g. `lualatex` is only allowed in configuration '`latex2pdfCommand`'. If used in configuration '`makeIndexCommand`' this causes this exception, because in that configuration, e.g `makeindex` is allowed.
`TSS09` `The diff directory '$diffDirectoryFile' should be a directory if it exists, but is not.`
The `$diffDirectoryFile` shall exist and be a directory. In it shall be stored the artifacts the actually created shall be compared with if `chkDiff` is set using the command `diffPdfCommand`. As the name suggests, currently only pdf-files are compared.
`TSS10` `Specified unregistered converter '$convStrProper' with invalid category '$catStr'; should be '…'.`
The converter `convName` is specified in the setting `<catCommand>` in the form `convName:notCat` with category `notCat` not coinciding with `cat` as required.

Table 7.4: The `BuildFailureException`s of the class `Settings`

Id. Message Explanation
`TMI01` `Cannot get stream to file '$fileName'.`

Stream to file within jar. This may be the manifest file, pom.properties or git.properties.
`TMI02` `Cannot load properties from file '$fileName'.`
Provided the stream to the file is ok, could not load property. This may occur for pom.properties or git.properties.
`TMI03` `IOException reading manifest.`
Provided the stream to the manifest file is ok, could not read completely.

Table 7.5: The `BuildFailureException`s of the class `MetaInfo`

Id. Message Explanation
`TFU01` `Cannot create destination directory '$targetDir'.`
This is mainly because of writing permissions.
`TFU04` `Cannot overwrite directory '$destFile'.`
Because this plugin shall not turn directories into regular files and vice versa. This failure indicates that another plugin/task disturbs this one.
`TFU06` `Cannot copy file '$srcFileName' to directory '$targetDir'.`
This is mainly because of writing permissions.

Table 7.6: The `BuildFailureException`s of the class `TexFileUtilsImpl`

Id. Message Explanation
`TLP01` `Artifact '$pdfFileAct' from '$texFile' could not be reproduced.`
Processing $texFile' yields $pdfFileAct which is not "alike" the stored version. Currently, that kind of check can be performed for `pdf` files only. Also the diff check is executed only if parameter $chkDiff described in Section 6.12 is set. Then the diff command $diffPdfCommand is performed to determine whether the artifacts are equivalent in the sense given by the diff command. The concrete meaning of that equivalence may range from strict equivalence to some kind of visual equivalence.
`TLP02` `No file '$pdfFileCmp' to compare with artifact from '$texFile'.`
The PDF file '$pdfFileCmp' expected for comparison with the PDF file created from '$texFile' does not exist. It is expected only if a diff check is configured according to $chkDiff described in Section 6.12. Currently, that kind of check can be performed for `pdf` files only.
`TLP03` `Failure while writing file '$fileName' or closing in-stream.`

Failure while performing goal `inj` while writing file '`$fileName`' or closing in-stream. The file is created from a template replacing parameter names by their actual values. A reason may be that the template cannot be read or its in-stream cannot be closed. The result is written into the latex source directory.

Table 7.7: The `BuildFailureException`s of the class `LatexProcessor`

FIXME: to be added.

7.2 Logging of warnings and errors

The rules for logging warnings and errors is, that the user must be notified, if something went wrong, but the run is not aborted, by a warning or an error. It is not required that for each detail going wrong, there is a separate notification, but the user must be sure, that all is ok, if no warning and no error occurs.

To decide whether it is an error or a warning to be logged, one has to distinguish, whether the problem occurs when running an external application or within internal code. In the first case, the decision whether it is an error or a warning is left to that application:

- If the application returns an error code other than 0, it is an error.
- If the application is expected to write a log file, but none is found, it is an error. The applications used here, return a nontrivial error code if no log file is written.
- The applications used here, writing a log file distinguish between error and warning. If a log file is written both are logged in the log file and can be distinguished by the form of the entry via pattern matching. If no error occurs, the return code is 0, even if warnings occur.
- If an application writes at least one error into the log file, this software logs an error.
- If an application writes no error into the log file but at least one warning, principally this software logs a warning. There may be parameters to switch off warnings partially or all of them, but there must be also a configuration of parameter values that allow logging all warnings.

If an application does not create the expected output file, this software logs an error. This may be because of an internal error as described above, but also

because of wrong parameters. So, e.g. `lualatex -v xxx.tex` does not create a pdf-file as expected.

Id. Message Explanation
`EEX01` `Running $command failed with return code $returnCode.` Compare with TEX01 in Table 7.3: Error execution means that there is even no valid return code.
`EEX02` `Running $command failed: No target file '$fileName' written.` FIXME
`EEX03` `Running $command failed: Target file '$fileName' is not updated.` The command `$command` is expected to write to the file '`$fileName`' but this file is not updated. This indicates an error executing `$command`.
`WEX04` `Cannot read target file '$fileName'; may be outdated.` FIXME
`WEX05` `Update control may emit false warnings.` FIXME
`EAP02` `Running $command failed: No log file '$logFileName' written.` The command `$command` is expected to write a log file '`$logFileName`' but no such file exists. This indicates an error executing `$command`.
`EAP01` `Running $command failed. Errors logged in '$logFileName'.` The command `$command` logged at least one error in the file '`$logFileName`', where more details can be found.
`WAP03` `Running $command emitted warnings logged in '$logFileName'.` The command `$command` logged at least one warning in the file '`$logFileName`', where more details can be found. Note that if `$command` is a latex processor, this warning comes only iff the parameter `$debugWarnings` is set. Note also that notifications on bad boxes are not counted as warnings here.
`WLP03` `Running $command created bad boxes logged in '$logFileName'.` Here, `$command` is a latex processor. It logged at least one bad box, overfull or underfull, horizontal or vertical in `$logFileName` where more details can be found. Note that this warning comes only iff the parameter `$debugBadBoxes` is set.
`WLP06` `Running $command found issues logged in '$logFileName'.` **This warning does no longer occur. The following is the original explanation:** Here, `$command` is a checker tool. Strictly speaking, unlike the other warnings here, this does not signify that running `$command` went wrong but uncovered an issue (warning/error/message) logged in a file.
`WLP05` `Use package splitidx without option split in $texFileName.` This indicates that an extended idx-file "`xxx-yy.idx`" has been found without `xxx.idx` or without according entry `\indexentry[yy]{…}{…}` in `xxx.idx`.
`WLP07` `Found both '$dviFile' and '$xdvFile'; convert the latter.`

This indicates that for conversion to PDF there are a DVI-file and a XDV-file which may come from mixed application of `xelatex` and another converter. In this case, the `$xdvFile` is converted.

Table 7.8: The errors and warnings on running a command

Id. Message Explanation

`WFU01 Cannot read directory '$dir'; build may be incomplete.`
FIXME
`WPP02 Cannot read tex file '$texFile'; may bear latex main file.`
FIXME
`WAP04 Cannot read log file '$logFileName'; may hide warnings/errors.`
FIXME
`WLP02 Cannot read log file '$logFileName'; $kind may require rerun.`
FIXME
`WLP04 Cannot read idx file '$idxFileName'; skip creation of index.`
FIXME
`WFU03 Cannot close '$closeable'.`
FIXME
`EFU05 Cannot delete file '$delFile'.`
 	
`EFU06 Cannot move file '$fromFile' to '$toFile'.`
FIXME

Table 7.9: The errors and warnings on files/streams

Id. Message Explanation

`WPP03 Skipped processing of files with suffixes $skipped.`
FIXME
`WPP04 Skip processing $srcFile: interpreted as target of $lmFile.`
FIXME
`WPP05 Included latex files which are not latex main files:`
`$includedNotMainFiles.`
In parameter `mainFilesIncluded` only latex main files shall be mentioned. The above message shows files specified which are not recognized as latex main files. This is also affected by parameter `patternLatexMainFile`.
`WPP06 Excluded latex files which are not latex main files:`

`$excludedNotMainFiles.` In parameter `mainFilesExcluded` only latex main files shall be mentioned. The above message shows files specified which are not recognized as latex main files. This is also affected by parameter `patternLatexMainFile`.
`WPP07 Included/Excluded latex main files not identified by their name: $inclExcl.` This indicates that there are different latex main files with the same name (of course in different directories) and that `$inclExcl` are those given in parameter `mainFilesIncluded` or `mainFilesExcluded`.
`WLP01 LaTeX requires rerun but maximum number $maxNumRerunsLatex reached.` FIXME
`ELP01 For command '$command' found unexpected return code $returnCode.` Here, `$command` is a checker tool. The return codes are determined by reverse engineering. So possibly `$returnCode` cannot be interpreted.
`ELP02 Checker '$command' logged an error in $clgFile.` Indicates that the checker found an error. Note that errors are warnings declared explicitly as errors. There is also the case that warnings are declared as simple messages and thus causes neither a warning nor an error.
`WLP08 Checker '$command' logged a warning in $clgFile.` Indicates that the checker found a warning. Implicitly it means that no error was found since this would cause EPL02. Note that warnings can be declared as simple messages and thus cause neither a warning nor an error.

Table 7.10: Miscellaneous errors and warnings

FIXME: to be added.

Chapter 8

Gaps

Only figures created with xfig and stored as files pdf and ptx may be integrated into a LaTeX document. This could be extended to a broader variety of export file formats. The problem is, that fig-files to not contain information on the export format. This has to be either given elsewhere in a config file or determined by pre-parsing the tex-files.

There is no support for pictures in Graphics Interchange Format, allows also animations (gif)-format but maybe a converter to png is all needed.

There is no proper make-mechanism taking dependencies into account. Thus all documents in all formats specified are remade, whether they changed or not.

Also, if more than one target is created from one LaTeX source, common steps are redone for each target. E.g. if pdf and html are created, pdf creation is done twice and if pdf, html, odt and docx are created, odt is done twice (once for odt second for docx) and pdf is done even trice: once for pdf itself, once for odt and once for docx.

Creating more than one index is supported only via package `splitidx` in conjunction with `SplitIndex`. There are the following packages also supporting multiple indices but not supported officially: `index` described in [Jon95], `amsmidx` described in [Bee07] and `imakeidx` described in [Gre16]. Note that the package `multind` is obsolete.

According to [Tal16], Section 1, there are three options to create a glossary, whereas this software supports option two only, which uses `makeindex`. Also, although the package `glossaries` itself supports multiple glossaries via the command

```
\newglossary[log-ext]{name}{in-ext}{out-ext}{title}[counter]
```

described in [Tal16], Section 12, this software only supports creating a single glossary.

Reading [Tal16], Section 15.1, the glossarystyle `index` seems to allow creating indices through the `glossaries` package making any index-package obsolete.

For development given the LaTeX main file `xxx.tex`, the files `xxx.pdf`, `xxx.pdf`, `xxx.synctex.gz` and `xxx.log` are vital. Thus it would be fine to have a goal which touches these files or to have a parameter to touch these prior to creation to avoid that these are cleaned up after the run. This is an alternative to setting parameter `cleanup` to `false`. On the other hand, goal `grp` creating graphics in conjunction with a development tool like `auctex` for `emacs`, allows to compile a latex main file in that tool and thus to access `xxx.log` and `xxx.pdf`.

There are lots of possible improvements to be done on the goal *check*.

The ant-task does not allow to create single formats, e.g. pdf selectively.

The ant-build is not completed: tests are not run and test runs are no prerequisite for installation.

This manual is not finished. To test the overall functionality of the maven-plugin and of the ant-task described here, this manual is created through plugin and task.

Support for djvu via pdf2djvu: pdf2djvu -o output_file input_file

pdf2dsc (ps with document structuring convention)

pdf2svg is not so useful.

pdftohtml -c is also not bad,

consider also pdftocairo for creation of tiff and ps and many others.

Chapter 9

Bugs

Seemingly, indices and glossaries based on page numbers (there seems to be an alternative to this), may be out of date with the current algorithm: First `lualatex` (or some other converter) is run to create the raw index. Then a sorting program like `makeindex` is called which creates the sorted, collected and formatted index. Then one `lualatex` run is required to include this index into the created pdf-file. A second `lualatex` run is required to write the index to the table of contents, as typically required. The problem with this procedure is, that the subsequent runs of `lualatex` change the raw index which requires rerunning `makeindex` and after that again `lualatex`.

One way to solve that problem is to use the package `imakeidx` (improved makeidx) instead of the traditional package `makeidx`. This offers also multiple indices, which is another gap to be filled. Seemingly, `imakeidx` does not support glossaries and so for these, another solution is required, although the problem is the same.

Packages `robustindex` and `robustglossaries` offer another solution. The advantage would be to have handled both index and glossary. Also support of hyperrefs within indices and glossaries seem to be expanded. On the other hand, the two packages seem experimental and seem to to play with package `hyperref`.

The current implementation is based on package `rerunfilecheck` which works for index but not for glossary.

Check whether `glossaries` option `autorun` makes sense. Seems to run the command `makeglossaries` after each latex run. But how to find out whether to rerun latex???

Pattern to identify latex main file: Documentation: shall not include the environment `documentclass/documentstyle` in an input. Also check `RequiresPackage` and check whether `(re)newcommand` is possible or makes sense.

Maybe there is a bug in the number of reruns: I think, makeglossaries is like bibtex needing two latex reruns and not like makeindex, which requires a single

rerun.

Since this software heavily relies on `rerunfilecheck`, maybe a warning if not used is a good idea.

Figures are missing in html output Formulae are missing in html output. Index is s missing in html output. Glossary occurs in the toc but is not numbered.

Did not find a way to add a numbered entry for the glossary into the table of contents.

The pattern `(! )` detects an error only `-no-file-line-error` (which is the default) is set but does not work with option `-file-line-error`. This yields

```
./manualLMP.tex:2500: Undefined control sequence.
l.2500 \bla
```

instead of

```
! Undefined control sequence.
l.2500 \bla
```

I ask myself how to detect this error in file line error mode!

Pattern matching is line-wise. This is inappropriate for patternLatexMainFile but also for further patterns like multiline-warnings.

Also there seems to be a bug in java's regex package, which leads to non-termination: pattern `(\s*)*xx` seems not to terminate.

A problem is also that the ending "`.svg`" may occur as a source and as a target file of htlatex. Thus `mvn latex:clr` tries to delete the targets of the svg-files, although these are not sources but themselves targets.

A way to solve this problem is, to apply the delete pattern to graphic source files and the files created. CAUTION: for svg, the files created by the latex run shall be taken into account. A warning shall be issued for each matching.

Target html: references to figures are missing. jpg and png-pictures oddly represented. With option svg: problem. Leave away, then at least the formula occurs. But then, from the mixed pictures only the text occur, whereas the pdf is still missing. Maybe htlatex still relies on eps-format. Table is very wide. Umlauts and sz maybe also not properly represented.

Still for target html: currently all aspects making problems are deactivated: Figures, index and glossary. For the index have a look at the log-file. These aspects must be re-integrated as soon as possible.

For html: run package tex4ht with option `info` to obtain further options and their descriptions. Also add a proper description into this manual.

For files `.directory` ("`.`" first), the separation of root and suffix does not work. Maybe the best to ignore files like that.

Target `txt`: seems as if index and glossary not up to date.

target `pdf`: Idea to run `makeglossaries` always prior to `lualatex`.

Maybe this is more a gap than a bug: support for dvi-creation should be provided separately.

For target dvi, neither png nor jpg-pictures are included. The other formats work with `$pdfViaDvi` set. Note that the postscript-files must be in the same directory as the dvi, probably because it includes them only by link.

For the other case, `$pdfViaDvi` unset, this requires some research.

Also for creation of the txt-format, `$pdfViaDvi` must be set.

FIXME: on bibliography, index and glossary

The application `chktex` does not necessarily return an error code if something goes wrong, e.g. reading `-l chktexrc`. Thus only in debug mode one can recognize the misbehavior. This knocks out detection of build failures.

Also I would like to replace the global `chktexrc` by a local version, via '`-g0 -l chktexrc.my`'. The problem is, that the file is interpreted relative to the working directory.

The application chktex has an option `-I` to specify, whether input files shall be read. If not, creation of graphics is immaterial. I can also imagine, that one wants to configure, whether graphics shall be created or not.

It may make sense to define in `chktexrc` another verbosity level with format allowing to decide whether there is a warning/error/message. Now I modified the levels that all but the headlines start with blank. This makes it easy in -v1 and in -v2 to detect warning/error/message at the beginning of a line, without the risk of false error because a message is logged on a text starting with the word "error".

Maybe this is not a bug but an inconsistency between auctex and local config: Running with the plugin, e.g. with `pdflatex`, we obtain

```
This is pdfTeX, Version 3.14159265-2.6-1.40.15 (TeX Live 2014) (preloaded format=pdflatex 2014.8.9)  30 JAN 2017 10:58
entering extended mode
 \write18 enabled.
 Source specials enabled.
 %&-line parsing enabled.
**test.tex
(./test.tex
```

whereas running from within Emacs with auctex we obtain

```
This is pdfTeX, Version 3.14159265-2.6-1.40.15 (TeX Live 2014) (preloaded format=pdflatex)
 restricted \write18 enabled.
entering extended mode
```

and also the behavior is slightly different, e.g. on file

```
\documentclass{article}
\begin{document}
 äö¿
\end{document}
```

The parameter `patternReRunLatex` treated in Section 6.4.3 needs more careful investigation. This is done a little bit in class `org.m2latex.core.Settings`.

Chapter 10

Tests

FIXME: this chapter describes the tests to be performed.

Missing are tests on logging, tests on various input formats, output formats, tests including several paths defined by invocation of auxiliary applications for index, glossary, …

Chapter 11

Bibliography

[Aa08] Ola Andersson and al. Scalable Vector Graphics (SVG) Tiny 1.2 Specification. Technical report, W3C, `https://www.w3.org/TR/SVG/`, 12 2008.

[Ars09] Donald Arseneau. *The import package*. asnd@triumf.ca, 3 2009. This manual corresponds to import v5.1, dated 23-Mar-2009.

[BB16] Javier Bezos and Johannes L. Braams. *Babel*, version 3.9r edition, 4 2016.

[Bee07] B. Beeton. *The amsmidx package*. American Mathematical Society, `https://www.ctan.org/pkg/amsmidx`, version 2.02 edition, 9 2007.

[BLC+14] J. Braams, L. Lamport, D. Carlisle, F. Mittelbach, R. Schöpf, A. Jeffrey, and C. Rowley. *Standard LaTeX 2ε packages makeidx and showidx*. LaTeX Project, `http://latex-project.org/bugs.html`, 9 2014.

[Car98] David Carlisle. *The longtable package*, v4.09 edition, 5 1998.

[Car14] David Carlisle. *The ifthen package*, v1.1c edition, 9 2014.

[Car16] D. P. Carlisle. *Packages in the 'graphics' bundle*. `https://www.ctan.org/pkg/graphicx`, 5 2016. The LaTeX3 Project.

[Col23] J. Collins. latexmk - generate latex document. available at `https://ctan.org/pkg/latexmk/?lang=en`, 4 2023.

[Cré11] J. Crémer. A very minimal introduction to TikZ. `https://cremeronline.com/LaTeX/minimaltikz.pdf`, 3 2011. Toulouse School of Economics jacques.cremer at tse-fr.eu.

[Da11] Erik Dahlström and al. Scalable Vector Graphics (SVG) 1.1 Specification. Technical report, W3C, https://www.w3.org/TR/SVG/, 8 2011.

[DHH02] David Duce, Ivan Herman, and Bob Hopgood. Svg tutorial. Technical report, Oxford Brookes University, W2C, 2002.

[Fea16] Simon Fear. *Publication quality tables in LaTeX*. 300A route de Meyrin, Meyrin, Switzerland, v1.618033 edition, 4 2016.

[Ghe19] Ovidiu Gheorghieş. *MetaUML: A Manual and Test Suite*, 2 2019.

[Grä96] George Grätzer. *Math into LaTeX*. Springer Science, New York, 1996.

[Gre16] E. Gregorio. *The package imakeidx*. https://www.ctan.org/pkg/imakeidx, v1.3e edition, 10 2016. Enrico.Gregorio@univr.it.

[Hec05] A. Heck. Learning MetaPost by doing, 2005. https://staff.fnwi.uva.nl/a.j.p.heck/Courses/mptut.pdf.

[HH13] T. Henderson and S. Hennig. *A Beginner's Guide to MetaPost for Creating High-Quality Graphics*, 6 2013. https://www.tug.org/docs/metapost/mpintro.pdf.

[HMH15] Jobst Hoffmann, Brooks Moses, and Carsten Heinz. *The Listings Package*. j.hoffmann(at)fh-aachen.de, v1.6 edition, 6 2015.

[Hob14] John D. Hobby. *MetaPost, a user's manual*, 5 2014. for version 2.00, https://www.tug.org/docs/metapost/mpman.pdf.

[Ilt12] Philip Ilten. *The svg Package*, v1.0 edition, 9 2012. philten@cern.ch.

[ISOa] International Organization for Standardization (ISO). *Document management - Electronic document file format for long-term preservation - Part 1: Use of PDF 1.4 (PDF/A-1)*.

[ISOb] International Organization for Standardization (ISO). *Document management - Electronic document file format for long-term preservation - Part 2: Use of ISO 32000-1 (PDF/A-2)*.

[ISO12] International Organization for Standardization (ISO). *Document management — Electronic document file format for long-term preservation — Part 3: Use of ISO 32000-1 with support for embedded files (PDF/A-3)*, 2012.

[JM15] Alan Jeffrey and Frank Mittelbach. *inputenc.sty*. The LaTeX project, http://latex-project.org/, v1.2c edition, 3 2015.

[Jon95] David M. Jones. *A new implementation of LaTeX's indexing commands*. https://www.ctan.org/tex-archive/macros/latex/contrib/index?lang=en, v4.1beta edition, 9 1995.

[Ker16] Uwe Kern. *Extending LaTeX's color facilities: the xcolor package*. www.ukern.de/tex/xcolor.html, xcolor@ukern.de, v2.12 edition, 5 2016.

[Koh16] M. Kohm. *Creating More Than One Index Using splitidx and SplitIndex*. https://www.ctan.org/pkg/splitindex?lang=en, v1.2c edition, 2 2016. komascript@gmx.info.

[Kwo88] C. Kwok. *EEPIC Extensions to epic and LaTeX Picture Environment Version 1.1*. Department of Electrical Engineering and Computer Science, University of California, Davis, California, 2 1988. https://www.ctan.org/pkg/eepic?lang=de.

[Lam87] Leslie Lamport. MakeIndex : An Index Processor For LaTeX, 2 1987.

[LRZ] MakeIndex - ein Indexprozessor füer LaTeX. https://www.lrz.de Menues: services, software, textverarbeitung, makeindex.

[Mar09] Nicolas Markey. Tame the BeaST - the B to X of BibTEX. manuscript, 10 2009. markey@lsv.ens-cachan.fr.

[MFL16] Frank Mittelbach, Robin Fairbairns, and Werner Lemberg. *LaTeX font encodings*. The LaTeX3Project Team, 2 2016.

[Mös98] Peter Mösgen. Makeindex Sachregister erstellen mit LaTeX. Katholische Universität Eichstätt Universitätsrechenzentrum, 5 1998.

[Obe16a] Heiko Oberdiek. *The bmpsize package*. heiko.oberdiek@googlemail.com, v1.7 edition, 5 2016.

[Obe16b] Heiko Oberdiek. *The rerunfilecheck package*, v1.8 edition, 5 2016.

[Obe16c] Heiko Oberdiek. *The transparent package*, v1.1 edition, 5 2016.

[Pat88] Oren Patashnik. BibTEXing. manuscript, 2 1988.

[PDF08] Adobe Systems Incorporated 2008. *Document management – Portable document format – Part 1: PDF 1.7*, 1 edition, 7 2008.

[Poo] Geoffrey M. Poore. PythonTEX Quick-start. https://github.com/gpoore/pythontex/blob/master/pythontex_quickstart/pythontex_quickstart.pdf.

[Poo15] Geoffrey M. Poore. PythonTeX: reproducible documents with LaTeX, Python, and more. *Computational Science & Discovery*, 8(1), 7 2015. doi:10.1088/1749-4699/8/1/014010.

[Poo17] Geoffrey M. Poore. PythonTEX Gallery. https://github.com/gpoore/pythontex/blob/master/pythontex_gallery/pythontex_gallery.pdf, 7 2017.

[Poo21] Geoffrey M. Poore. *The pythontex package*. gpoore at gmail.com, github.com/gpoore/pythontex, v1.8 edition, 6 2021.

[Rei16] E. Reißner. The xfig file format for xfig 3.2. see http://www.simuline.eu/LatexMavenPlugin/xfig/xfigFormat.pdf, 12 2016.

[Rei17] E. Reißner. The DVI-format and the program DVItype. http://www.simuline.eu/LatexMavenPlugin/dvi/dviFormat.pdf, 1 2017.

[Rei22] E. Reißner. Files, errors and warnings of pythontex 0.18. available at http://www.simuline.eu/LatexMavenPlugin/pythontex/pythontexInOut.pdf, 7 2022.

[Rei23] E. Reißner. Special and common aspects of pdf/dvi/xdvi generators. http://www.simuline.eu/LatexMavenPlugin/latex/latexEngines.pdf, 3 2023.

[RO22] Sebastian Rahtz and Heiko Oberdiek. *Hypertext marks in LaTeX: a manual for hyperref*, 2 2022.

[Sch11] Ulrich Michael Schwarz. *The nag package*. absatzen, http://absatzen.de/, ulmi@absatzen.de, 11 2011. corresponds to nag 0.7, dated 2011/11/19.

[Sch16] R. Schlicht. *The microtype package*. w.m.l@gmx.net, v2.6a edition, 5 2016.

[Sio17] Laurens Sion. *The pdfprivacy package*. Princeton University, laurens@sion.info, v1.0 edition, 12 2017.

[SMCR15] Walter Schmidt, Frank Mittelbach, David Carlisle, and Chris Rowley. *The fix-cm package*. The LaTeX Project Team, v1.1t edition, 1 2015.

[SU06] A. Simonic and S. Ulrich. *srcltx.sty · srctex.sty*. stefanulrich@users.sourceforge.net, v1.6 edition, 11 2006.

[Sza07] Péter Szabó. *The anyfontsize package*. pts@fazekas.hu, 2 2007.

[TAK+14] Kresten Krab Thorup, Per Abrahamsen, David Kastrup, et al. *AUCTeX A sophisticated TEX environment for Emacs*. Free Software Foundation, Inc., version 11.88 edition, 10 2014.

[Tal16] N. L.C. Talbot. *User Manual for glossaries.sty v4.26*. dickimaw-books, http://www.dickimaw-books.com/, 10 2016.

[Tal21] N. L.C. Talbot. The glossaries package v4.49: a guide for beginners. https://www.ctan.org/pkg/glossaries?lang=de, 11 2021.

[Tan15] T. Tantau. *TikZ and PGF Manual for Version 3.0.1a*. Institut für Theoretische Informatik, Universität zu Lübeck, Lübeck, Germany, 8 2015. http://sourceforge.net/projects/pgf.

[Tea00] The LaTeX3Project Team. *LaTeX2ε font selection*, 9 2000.

[Tea22] The LaTeX Project Team. *The iftex package*. https://github.com/latex3/iftex, v1.0f edition, 2 2022.

[Thi22] Jens T. Berger Thielemann. *ChkTEX v1.7.8*. Jens Berger, Spektrumvn. 4, N-0666 Oslo, jensthi@ifi.uio.no, 10 2022. New maintainer: Ivan Andrus, darthandrus@gmail.com.

[TW12] Julian Ohrt Thomas Willwacher. Tikzedit a semi-graphical Tikz editor. `http://www.tikzedt.org/`, t.willwacher@gmail.com, 2012.

[Ume10] Hideo Umeki. *The geometry package*. latexgeometry@gmail.com, v5.6 edition, 9 2010.

[Wic99] Mark A. Wicks. *Dvipdfm User's Manual*, 9 1999. Version 0.12.4b.

[WK20] Thomas Williams and Colin Kelley. *gnuplot 5.4 – An Interactive Plotting Program*. `http://sourceforge.net/projects/gnuplot`, 7 2020. Version 5.4.

[WP10] P. Wilson and H. Press. *The tocbibind package*. latex-project, `https://www.ctan.org/pkg/tocbibind?lang=de`, v1.5k edition, 10 2010.

[Zan10] Timothy Van Zandt. *The 'fancyvrb' package Fancy Verbatims in LaTeX*. Princeton University, tvz@Princeton.EDU, v2.8 edition, 5 2010.

Chapter 12

General Index

Chapter 13

LaTeX Packages