Manual for the `latex-maven-plugin` and for an according ant-task Version 2.1

Ernst Reissner (rei3ner@arcor.de)

September 20, 2024

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 PDF (Portable Document Format)-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 DVI (DeVice Independent; traditional output format of LaTeX engines, today widely replaced by PDF) (device independent) described in [Rei17] has been used and still creation of external formats like HTML (HyperText Markup Language), ODT (Open Document Text) and DOCX (current document format for MS Word) are based on an intermediate DVI-file. It is much more lightweight than PDF specified in [PDF08], and the newer [ISO20].

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, only the so-called LaTeX main files must be compiled using a so called LaTeX engine, other LaTeX files occur only as input for the `\input` command. 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 and thus need special support by this software. If there is some format needed but not yet provided, please write an email to the author.

Many graphic file formats require preprocessing, i.e. the according files must be processed 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, the target formats include the web-formats HTML and XHTML (eXtensible HyperText Markup Language), 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 FIG (native file format for xfig)-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 MP (MetaPost: input format for the graphic program `mpost`)-files into EPS (Encapsulated PostScript)-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 SVG (Scalable Vector Graphics)-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 LaTeX engines like `lualatex`, `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 [Tal24b]. Note that `makeglossaries` either invokes `makeindex` or `xindy`, depending on the parametrization of the package `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`.

  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 SGML (Standard Generalized Markup Language) and XML (eXtensible Markup Language), 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]

anyfontsize described in [Sza07] to allow arbitrary font sizes, eliminating certain warnings. An alternative may be `fix-cm` described in [SMCR15].

Almost required are

- `rerunfilecheck` described in [Obe22] which writes additional rerun warnings to the log file if some auxiliary files have changed. This software partially relies on these warnings to control rerun the LaTeX engine. Although `rerunfilecheck` is intended to detect also rerunning auxiliary programs, this does not work properly and so this software bases reruns on internal algorithms.
- `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 [Obe16b], 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 [Tal24b] with tutorial [Tal24a] 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 [BB24] 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>2.1</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 3.6, 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>2.1</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 3.5. 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 settings of this plugin. Note that the execution of the goal `inj` has its own configuration, which allows a single parameter, `injections`. Listing 2.4 gives a recommended parameter value, although not the default.

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 3.6, 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. Since cleanup occurs in the course of the build and not with goal `clr` the parameter `cleanUp` is given in the general settings. The goal `clr` cannot be configured.

Finally, it is recommended to add a check of the versions of the programs called *converters* used 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 configuration `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 3.2. 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 3.6.

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`.

It is planned to perform a version check in first usage of a tool, except tools in the environment, i.e. build tools and programming languages. This avoids check of tools which are not needed. Also, for the generic user, no execution for goal `vrs` is needed any more; by need it can be invoked from the command line as `mvn latex:vrs`. Still the developer of this software will continue to specify that execution.

Note that in Listing 2.4 the section `configuration` which is not part of an execution contains an empty configuration and is thus as much as empty. It can thus 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%`

```
<plugin>
  <groupId>eu.simuline.m2latex</groupId>
  <artifactId>latex-maven-plugin</artifactId>
  <version>2.1</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-->
      <configuration>
      <injections>latexmkrc,chktexrc,header</injections>
    </configuration>
    </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 jar for the tasks, `latex-maven-plugin-2.1-antTask.jar`, must be downloaded manually from

`https://www.simuline.eu/RepositoryMaven/eu/simuline/m2latex/latex-maven-plugin/2.1%`

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:

- 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-2.1-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-2.1-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 3.8 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 3.6.

In contrast, Section 3.7 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 and their directories

Source files are files contributing to creating documentation from LaTeX-files in the build process which are not themselves created in the build process. They 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/ant-project. This structure complies with conventions in maven-projects.

Note that, against the convention of maven-projects, the TeX source directory may contain also files created during the build process. By default, after the build

process is finished, they are removed again. For some background on this see Section 3.1.3.

Source files may be TEX files treated in Section 3.1.1 and various kinds of graphic files described in Section 3.1.2, but may include also

- verbatim text embedded into TEX files with `verbatim`,
- BIB files typically describing a bibliography, or, not yet supported, a glossary or that like,
- program files, either included as a listing by package `listings` or executed via the package `pythontex`.

3.1.1 LaTeX main files and other latex files

The TEX files are special in that only part of them is processed explicitly invoking a compiler like `lualatex` on them, part is just included via `\input` or `\include`. The LaTeX-files to be compiled top level, 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: it is intended to be input in another TEX file, in this case `manualLMP.tex`.

LaTeX main files are detected automatically by fitting the regular expression `patternLatexMainFile` described in detail in Table 6.1 on page 122, and in the reference given there, whereas the description here is quite high level.

As a first approximation, a LaTeX main file is one invoking the command `\documentclass` or the outdated `\documentstyle`, both specifying the document class. It must be excluded that the pattern matches a textual occurrence of `\documentclass`, which just occurs because the document is on LaTeX and mentions the command `\documentclass`. This is quite easy, since there is little allowed in TEX files preceding these commands.

Consequently, the pattern matches the region from the start of the file to and including the `\documentclass` or `\documentstyle` command. This starting segment of a LaTeX main file is called the *opening*.

Here a word of warning is at place: if a TEX file does not fit the pattern, it is not interpreted as a LaTeX main file without further warning. So check whether the file under consideration is processed if

- it is built for the first time or
- its opening is changed or
- the parameter `patternLatexMainFile` is changed

If you distrust the recognition mechanism via pattern matching altogether, you can explicitly specify each LaTeX main file in the parameter `mainFilesIncluded` described in Table 6.1 on page 122. This is safe because if a file specified in `mainFilesIncluded` that like is not a LaTeX main file according to the pattern `patternLatexMainFile`, a warning is emitted. That way one can check whether the pattern is matched. We could have decided that these files are compiled with or without warning, but this would lead to a technique is that it is inconvenient and not well maintainable.

On openings neglecting (magic) comments

This section contains material both specific for supported document classes and general information but magic comments are deferred to Section 3.1.1.

This software is tested for document classes

- `article` and `book` which are built-in,
- `beamer` for presentations as described in [TWM23],
- `leaflet` creating leaflets as explained in [SGNS20] and in [GNS20],
- `scrlttr2` for letters described in [Koh23], Part1, Chapter 4,
- and `minimal` for quite special uses (did not find real documentation).

Note that `scrlttr2` replaces the built-in `letter` which is not recommended. In fact, is the only KOMA class this software is tested for. The attentive reader may realize that the built-in document class `report` is not mentioned. It shall work but is currently not tested.

Nevertheless, the pattern `patternLatexMainFile` matches all possible document classes. Typically, the document class is loaded with options. The most frequent class may be `article` followed by `report` and `book`. For these, we suggest something like

```
\documentclass[a4paper,12pt,english]{article}
```

where `a4paper` is a setting, typical for Europe or, not the US. The default font size is `10pt` and sometimes it makes sense to increase this. For documents which are solely in English no language setting is required, except if loading the package `babel`, else the hyphenation patterns get lost. Since we recommend inputting the header file `header.tex` described in Section 3.5.2 which in turn loads `babel` and by the way also `csquotes`, a language setting is mandatory. It is possible specifying the language when loading `babel` as an option like so `\usepackage[english]{babel}`, but it is recommended to specify the language as an option of the document class,

in order to make it available for various packages, besides `babel` also for `csquotes`. If a document has more than one language, specify all of them, the last the one the document starts with, but [BB24] Sections 1.7 and 1.8 show how to change language, here temporarily into German, which requires specifying `german` in front of `english` in the document class. Note the quotes and the correct hyphenation in the following paragraph.

> Sie las den Artikel „Chancen für eine diplomatische Lösung“ in der „Wochenpost“, während er es sich nicht nehmen ließ, sich Thomas Manns Novelle „Der Tod in Venedig“ zu Gemüte zu führen.

To obtain correct quotes in the above paragraph, package `csquotes` must be loaded with option `autostyle`. Since `csquotes` is loaded in `header.tex` given by an injection as described in Section 3.5, this option must be passed to `csquotes` before loading the document class, e.g. via

```
\PassOptionsToPackage{autostyle}{csquotes}
```

This is the technique to pass options to packages in general.

For documents of class `minimal` there are no requirements imposed. No checks and no PDF-info. For all other document classes, it is recommended to load `nag` before `\documentclass` by

```
\RequirePackage[l2tabu, orthodox]{nag}
```

Thus the pattern `patternLatexMainFile` allows `\RequirePackage` with its options preceding `\documentclass`.

There are cases, where one and the same document comes in two flavors both of which must be built. As an example, consider a document with a confidential variant and with a non-confidential variant. To define these, the declaration of the document class must be preceded by the following kind of code:

```
\RequirePackage{etoolbox}
\newbool{isConfidential}
\setbool{isConfidential}{true}
```

The new thing is defining and setting a boolean via `\newbool` and `\setbool`, respectively. It is a good idea to set a watermark via

```
\ifbool{isConfidential}{%
  \usepackage{draftwatermark}
  \SetWatermarkText{Confidential}%
}{%
% no watermark text
}
```

The same technique differentiating between confidential and public may be used to define a lecture with and without solution or any other kind of variant.

Documents of class `leaflet` resemble `article`s, except special options like `notumble`.

```
\documentclass[a4paper,notumble,10pt,english]{leaflet}% 12pt,notumble
```

The same is true for letters of type `scrlttr2`, except for the special specification of font size and versioning of the class:

```
\documentclass[english,german,a4paper,fontsize=10pt,version=last]{scrlttr2}
...
\input{header.tex}
\LoadLetterOption{DIN}
```

Observe that after inputting `header.tex` which loads `geometry`, various pseudo lengths have to be re-adjusted. This is done by loading the letter option. Without it may happen, that the text does not reach until the bottom of the frame.

What is special for beamer presentations is, that in general two documents with the same identifier, e.g. title are created, the proper presentation, e.g.

```
\RequirePackage[l2tabu, orthodox]{nag}
\PassOptionsToPackage{colorlinks,linkcolor=blue,urlcolor=blue,citecolor=blue,destlabel}{hyperref}

\documentclass[10pt,english]{beamer}
\mode<presentation>%

\input{useBeamer}
```

and the corresponding handout

```
\RequirePackage[l2tabu, orthodox]{nag}

\documentclass[a4paper]{article}
\usepackage{beamerarticle}
\input{useBeamer}
```

both including the same piece of code which is included from file `useBeamer.tex`. The author recommends to stick to this convention. As an example document may serve [Rei23a] which is a presentation of this software including the handout and illustrates the use of the beamer class. Observe, that both documents use `header.tex` injected as described in Section 3.5.2 loading various packages. The beamer class is special in that it loads the `hyperref` package itself. To avoid option clash with `header.tex`, for document class `beamer` option `destlabel` must be passed to the package. Maybe it is a matter of taste, but `beamer` tends to make links invisible. To force loading options specifying colors for links and `destlabel`, use `\PassOptionsToPackage` as shown above.

All documents but beamer documents must specify the paper size globally via `\documentclass`. Beamer documents may specify accordingly `aspectratio`. All this must be allowed for LaTeX main files.

Magic comments

It also makes sense to allow comments also in openings, i.e. text from unescaped `%` to the end of the line, and also magic comments. A magic comment, as all comments, is ignored by the LaTeX compilers but give hints to more high level tools like IDEs or build tools like this LaTeX builder. It is the mechanism to treat a document in a specific way so magic comments override the general settings.

Typically, a magic comment comprises a whole line and starts with `% !`, maybe followed by an identifier of the tool it refers to or by an identifier referring to TEX files in general. For example latex workshop and TeXshop support the magic comment `% !TEX root` and this must be essentially in the first line. The magic comments specific for this tool may be preceded by general magic comments and start with `% !LMP` which is short for "latex maven plugin". This is not fully correct but easy to remember.

This LaTeX builder is designed to cooperate with other tools. The magic comments of the other tools as described in various places in Section 3.6 on document development and in particular in Section 3.6.2. Thus, if appropriate, also magic comments of other tools are read, except those of AUCTeX, because AUCTeX places magic comments at the end of file, forcing this software to read all the file if it accessed AUCTeX magic comments also. All other tools including latex workshop for VS Code support a subset of what is defined by TeXstudio. From all magic comments in the context of signifying LaTeX main files only `program` and `root` are relevant. If a root is given, then the file is no LaTeX main file and, provided the feature is used, also the converse is true. Since this software shall not rely on further tooling, it does not use `root`. All in all, among the general magic comments only `% !TEX program=…` is read. It can occur more than once, but the first occurrence is what counts; the others are ignored silently. Note that the magic comment `% !TEX program=…` overrides the setting `latex2pdfCommand` for creating PDF files and related, specified in Table 6.4 on page 134, but not the `tex4htCommand` from Table 6.10 on page 143.

After the general magic comments of the form `% !TEX …` come the ones specific for this LaTeX builder. They take the form `% !LMP …`. Like the general magic comments, the specific ones are all optional, but in contrast, they come in a fixed order without repetition.

What follows is a full range of magic comments:

```
% !TEX program=lualatex
% !LMP chkDiff
```

```
% !LMP latexmk
% !LMP targets=chk,pdf,html

\documentclass[a4paper]{article}
```

Section 6.2.1 describes the meaning of the individual comments in the course of explaining the pattern `patternLatexMainFile`. Note that there the names of the magic comments is given, whereas the above listing refers to the content, but it is easy to identify the according magic comments. The relation of the magic comments is described in the following.

The magic comments may come only in the ordering given in the above listing, but each of them is optional. They can be freely combined, but note that `chkDiff` and `latexmkMagic` apply to creation and check PDF files only. So, for `targets=pdf,html`, these magic comments apply to target `pdf`, but not to `html`. For `targets=chk,html` it even takes no effect at all without issuing a warning. As explained above, `program` affects only the targets `pdf` and `dvi` including also XDV files.

Note that documents of the classes `beamer`, `leaflet` and `scrlttr2` can essentially only be compiled into a PDF, and maybe further to a TXT file. In addition, to targets and goals `pdf` and `txt`, it can be checked with target or goal `chk`. Other targets are skipped, and a message is displayed. The relations are configurable through settings `targets` and `docClassesToTargets` both in Table 6.1 on page 122. Also, if a document class occurs, which is not registered in `docClassesToTargets`, a warning `WLP09` described in Table 7.10 on page 164 is displayed.

3.1.2 Source graphic files

The great bulk of file types occurring as sources, are graphic files in various formats. Note that this section is not about intermediate file types like PDF or MPS used to include the original file types into the target.

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 [Tan23].

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`, the latter is the default. It is not the package but the driver which decides on the support of graphic formats. The dvipdfmx user manual, [Tea20], Section

3.1.1 lists the allowed formats MetaPost (i.e. MPS (metapost's postscript like output including text)), postscript (i.e. EPS), PDF, JPG (Graphics format developed by the Joint Photographic Experts Group) including jpeg2000 and PNG (Portable Network Graphics).

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 [WK23], 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 [Hob24] 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 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.

3.1.3 Created files in the TeX source directory

Note that against maven convention and 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

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

intermediate files, deleting them again by default after the build process. 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 3.6 an in particular in Section 3.6.2.

The downside is that a file residing in the TEX source directory risks being overwritten or deleted 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.
- For LaTeX main files more general files are created, but they all must match those in pattern `patternCreatedFromLatexMain` described in Table 6.1 on page 122. So it is save to add files not matching this pattern.

Note Section 3.5 which is on goal `inj` injecting diles in the TEX source directory and Section 3.6.4 on goal `grp` processing graphic files which creates intermediate files therein also.

To get rid of intermediate files, there is a separate goal `clr` described in Section 3.6.5.

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>:<targets>`: The source files are in `latex`-format and the `targets` are read from the *configuration* in the pom (*configuration* is what `cfg` stands for) which is illustrated in Listing 2.3. For a detailed description of the setting `targets` see Table 6.1 on page 122. Here only an overview is given.

By default, the targets configured are `chk`, `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 DOC (outdated document format for MS Word) format.

```
<!-- create html and pdf and other formats from latex -->
<plugin>
  <groupId>eu.simuline.m2latex/groupId>
  <artifactId>latex-maven-plugin</artifactId>
  <version>2.1</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 engines `lualatex` and `pdflatex`, whereas `xelatex` creates the XDV (extended DVI) format for target `dvi`.

Somehow special is the target `chk` which is mere checking by invoking `chktex` 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 3.6.3.

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

Although the possible targets can be configured globally via the setting `targets`, the possible targets may depend on the document class of the LaTeX main file. At time of this writing, all document classes in preferred usage as defined in Chapter 10 support all targets with obvious exceptions: Besides checking (target `chk`) for obvious reasons the classes `beamer`, `leaflet` and the letter class `scrlttr2` directly support only target `pdf` and because texts are created from PDF files, also target `txt`. The mapping from document classes to allowed targets is given in setting `docClassesToTargets` given in Table 6.1 on page 122. This parameter restricts the targets given in parameter `targets`. As explained in detail in Section 3.1.1, if a document class cannot be identified by the command `documentclass` or the

outdated `documentstyle`, it can be specified by a magic comment directly.

Finally, the targets can be specified individually for each LaTeX main file using a magic comment as described in Section 3.1.1. A target specification in a magic comment overwrites all settings in `targets` and in `docClassesToTargets`. If a magic comment specifies the targets directly, the document class need not be known. In particular, a magic comment only specifying targets identifies already a LaTeX main file as specified in Section 3.1.1.

As a magic comment can be used to specify the target formats for a LaTeX main file individually, Section 3.1.1 shows how to specify the LaTeX engine to be used for this file overwriting the general setting `latex2pdfCommand` in the pom given in Table 6.4 on page 134.

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 3.6.5.

3.3 Checking versions of converters

The goal `vrs` is to display meta information, above all version information:

```
mvn latex:vrs
```

displays something like what is displayed in Listing 3.2. 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 122. Excluded converters are prevented from being used: if tried, Exception TSS07 described in Table 7.4 on page 159 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 142: `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.

3.4 Logging of errors and warnings

It is a design goal to notify the user of formal deficiencies of the created documents by tight logging of errors and warnings to give high confidence in the formal quality of the result in the absence of errors and warnings. Logging is devoted all of Chapter 7. Of particular intestest is Table 7.8 on failures running a tool on page 162. We highlight

- Check of return codes
- absent/outdated target files or log files
- log files containing errors and warnings (provided by many but not all tools). Among those notification of bad boxes.

3.5 Injection of files

The goal `inj` is to inject files into the working directory `texSrcDirectory`, by default in maven lifecycle phase `valiate` or from command line in the root directory. The injected files are in general adapted to the current configuration of the plugin.

Note that each of these files is written only if it is guaranteed that only files written by this plugin are overwritten. This is the case, if no file is overwritten at all or if the file to be overwritten is recognized to start with a comment indicating

```
[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]            mvn:                 '3.9.4'in[3.9.1;3.9.4]
[INFO]            ant:                 '1.10.14'in[1.10.12;1.10.14]
[INFO]            java:                '17.0.9'in[17.0.9]
[INFO]            python:              '3.11.6'in[3.11.6]
[INFO]            perl:                '5.38.2'in[5.38.2]
[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:           '23.11.0'in[21.04.0;23.11.0]
[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:             '23.11.0'in[22.01.0;23.11.0]
[INFO]            exiftool:            '12.71'in[12.39;12.71]
[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] WMI02:  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 10'in[5.4 patchlevel 0;5.4 patchlevel 10]
[INFO]            inkscape:            '1.3.2'in[1.0.2;1.3.2]
[INFO]            fig2dev:             '3.2.9'in[3.2.7b;3.2.9]
[INFO] tools excluded:
[INFO] upmendex, xindy
[INFO] tools not found:
[INFO] latexmk
[INFO] ------------------------------------------------------------------------
```

Listing 3.2: Output of goal `latex:vrs`

that this file is written by this plugin. Of course the guarantee holds only if the headline does not tell a lie.

If the headline 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 by goal `inj` and by the way not erased by goal `clr` as described in Section 3.6.5. In case of such a doubt, a warning is displayed.

That way, injected files written by the plugin can be updated each run, which is necessary to keep them synchronized with the configuration of this plugin, but according files written e.g. by the user are protected.

A first description of the goal `inj` is given by

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

which yields a list of files which can be injected. Note the distinction between the injection, which is the act of injecting and the according file which is injected.

The set of injections can be divided into the following categories according to the function of the files injected:

- Configuration files for `latexmk` and `chktex`. These are hidden files and form the default. In particular, the configuration file of `latexmk` is adapted to the configuration of this plugin, to ensure that the results are the same whether created by `latexmk` or by this plugin.

- Header files are intended to be included in TEX files. They load packages and provide commands. In general, header files are designed to run on all usual LaTeX compilers, with various document classes and take creation of PDF into account but also of other formats like HTML and also of DVI/XDV which is an important intermediate format.

  The packages are loaded with minimum options, but these can be modified outside the headers by `\PassOptionsToPackage` as described in Section 3.1.1.

- Script files which are intended to run by the user supporting the automatic build process "from outside" above all in the course of document development. Thus, usually, their injection is triggered selectively from the command line as described below. In contrast to the files in the other categories, these are executable.

Table 3.1 shows the possible injections and the ones really to be performed are given in the configuration `injections`. This configuration is described in Section 6.3 on page 127. It is a comma separated list and the default is `latexmkrc,chktexrc`, representing the configuration files.

Name	File	explanation
configuration files		
latexmkrc	`.latexmkrc`	config file for `latexmk`
chktexrc	`.chktexrc`	config file for `chktex`
header files		
header	`header.tex`	fundamental
headerGrp	`headerGrp.tex`	for graphics
headerSuppressMetaPDF	`headerSuppressMetaPDF.tex`	to control PDF meta-info
shell scripts		
vscodeExt	`instVScode4tex.sh`	installs VS Code extensions
ntlatex	`ntlatex`	timeless LaTeX compiler
vmdiff	`vmdiff`	special diff tool for PDF files
pythontexW	`pythontexW`	surrogate for `pythontex`
depythontexW	`depythontexW`	surrogate for `depythontex`

Table 3.1: Overview over all injections

As described in Section 2.2.3, by default the goal `inj` is tied to the maven phase `validate`, an early phase preparing the proper build process, because the injected files are a prerequisite for building. Then the files are injected in the TEX root directory `texSrcDirectory`.

On the other hand, injections can be also invoked by command line via `mvn latex:inj` with the default injections or, with given list of injections, e.g.

```
mvn latex:inj -Dlatex.injections=vscodeExt,ntlatex,vmdiff
```

In fact, injection from command line is typically used for scripts, whereas the others files are injected during the build process in phase `validate`. Of course maven is invoked from the project root and there also the prescribed files are injected.

Note that the folder where cleanup of injections with `mvn clean` is done, depends also on whether `-Dlatex.injections=…` is specified, but the value is irrelevant as long as it is valid.

In the sequel, all these injections are described in detail separately, but in fact they are all related. For example, `header.tex` handles the possible configurations reflected in `.latexmkrc`. It provides packages used in `headerGrp.tex` and provides commands to exclude checking by `chktex` controlled by `.chktexrc`. The header `header.tex` is very crucial for example controlling and guaranteeing rerun of the LaTeX engine by including package `rerunfilecheck`. Its presence makes the results uniform and is a cornerstone for quality guarantees. As said above, the default for injections is `latexmkrc,chktexrc`, but it is advisable to use `latexmkrc,chktexrc,header`.

Now let us treat the injections individually.

3.5.1 The configuration files `.latexmkrc` and `.chktexrc`

For document development the tool `latexmk` is a valuable build tool. Also, a linter like `chktex` is helpful both for end control and for document development.

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`. The configuration files determine the behavior of the two tools without further options. The user is kindly asked to help to improve these files, in particular `.chktexrc`.

Ideally, the injected `.latexmkrc` is adapted to the current settings of this plugin and so invoking `latexmk` invoked with its configuration file behaves like this latex plugin. Currently, not all possible settings of this plugin are taken into account in the `.latexmkrc`, but the magic comments in the source files described in Section 3.1.1 are read and taken into account as far as this makes sense.

For default settings, maybe partially overwritten by magic comments, this maven plugin and `latexmk` create the same target files. This is true even for this manual. In particular, the graphic formats described in Chapter 4 are supported. So are bibliographies, indices and glossaries and also material computed by `pythontex` as described in Chapter 5, but without reflecting all options and patters to supervise log files. Also, reproducibility check is supported including magic comments and all parameters.

At time of this writing, `.latexmkrc` works for various LaTeX generators but supports target `pdf` only, although in the long run also `chk` and `dvi` could be useful. Still, creation of PDF files is supported in both variants, via DVI/XDV or directly. Compilation via `latexmk` is based on code in `.latexmkrc` and this mimics a wide range of functionality offered by this tool. Among these also build for reproducibility checks as described in Section 5.8

A sensible config file `.chktexrc` mainly depends on the packages loaded. In Section 3.5.2 we suggest injecting also a header file `header.tex` loading packages. The config file `.chktexrc` is adapted to the header file `header.tex`.

Observe, that due to an incompatibility between tool `latexmk` and package `listings`, this manual can only be compiled with `latexmk` if `listings` is not only loaded but also patched as done by `header.tex` and described in Section 3.5.2. With that patch, this software yields the same resulting PDF file as compilation with `latexmk`.

Currently, `.chktexrc` serves only to suppress warnings, mainly on material which is the argument of commands or the content of an environment. What is really needed depends on the packages loaded and on the commands and environments defined in addition.

Since basic packages are loaded and basic commands are defined in the injected `header.tex` which is described in Section 3.5.2, it makes sense, to syn-

chronize `.chktex` and `header.tex`. As an example, `header.tex` loads package `listings` which provides the environment `lstlisting` and the command `\listinputlisting`. The content of both shall not be subject to checks via `chktex` and must thus be excluded in `.chktexrc`.

All this together illustrates why it is recommended to inject besides the default `.chktexrc` and `.latexmkrc` also `header.tex`.

As described in Sections 3.6.2 and 3.6.3, both tools `chktex` and `latexmk` are invoked directly by the user in the course of document development, but they may be invoked by this LaTeX builder in the course of a regular build, i.e. for maven goal `cfg` also. So their respective configuration files must be injected in the maven build process before the LaTeX build tools are invoked, i.e. prior to the phase `site`. Thus, goal `inj` has default phase `validate`.

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` which is described in [Col23], Section "LATEXMK OPTIONS AND ARGUMENTS ON COMMAND LINE".

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, [Thi22] does not tell about the ordering in which the configuration file given by the option `-l` is read in.

For sake of reproducibility, we recommend restricting to the global configuration file which is tied to the installation and to a local file, specific to the latex source directory which shall be valid for all LaTeX main files in that directory.

Goal `inj` injects the configuration files `.latexmkrc`, `.chktexrc` and further files, all in the latex source directory. It is natural to use each as the local configuration file.

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 instead of `.chktexrc`. Uniformity with respect to the operating systems can be realized with a link to `chktexrc` named `.chktexrc`. That way independent of the operating system, the configuration files `.latexmkrc` and `.chktexrc` are sufficient.

It is important that there is a unique central configuration file applying to all LaTeX main files. There is the choice between at least two mechanisms to ensure this: Either `latexmk` and `chktex` are invoked with options `-r` and `-l`, respectively, specifying the configuration file explicitly or for each folder containing a LaTeX main file there must be a link named `.latexmkrc` and `.chktexrc`, respectively, to the according central configuration file.

We recommend using links because then `latexmk` and `chktex` can be used on the command line without further options. This is convenient for the user when invoking the tools directly which is the typical usage for document development.

3.5.2 A generic header file `header.tex`

It is observed that the headers of various LaTeX files are quite similar. In particular the packages loaded have a huge overlap and at the same time, although rare, exotic packages tend to be 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 format into some log file.

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`.

Some loaded packages are also patched. The patch for package `listings` is given in Listing 3.3. It applies only if `listings` is loaded prior input of `header.tex`. One modification is, redefinition of `\lstlistoflistings` to make the list of listings occur in the table of contents and to rename the title so that it fits other lists as the list of figures. The other point is modifying `listings`' output to make it digestible for `latexmk`. For details see Section 3.6.2.

The other package patched is `luamplib`; the patch is given in Listing 3.4. It applies only if `luamplib` is loaded prior input of `header.tex`. As discussed in Section 4.5, `luamplib` is available for `lualatex` only. It provides an environment `mplibcode` to enclose literal MetaPost. The enhancement is an additional command `\inputmpcode` which allows including MetaPost files. This functionality is analogous to package `listings` which allows both literal listings and loading listings from files. MetaPost code within a LaTeX document typically disturbs syntax highlighting of both, enclosing code and included code.

Besides loading packages it also sets `synctex` which is crucial for synchronizing TEX files and according PDF files via forward search and backward search as described in Section 3.6.1.

It also provides the command `\setMinorVersionPdf` to set the minor version of the PDF file created. This is mostly needed because some graphic tool creates PDF files with a newer PDF version than the LaTeX distribution does. Setting the version high enough, avoids an according warning `WAP03` listed in Table 7.8 on page 162. The warning pattern is described in Section 6.5.2 in detail. As described in [MF23], Section 2 the recommended way to set major and minor version of the PDF output like in `\DocumentMetadata{pdfversion=1.7}`, but due to a bug, any

```
% this indirection is needed because \makeatletter, \makeatother and \xpatchcmd
% don't work inside an argument as discussed in
% https://tex.stackexchange.com/questions/719158/
% does-ifpackageloadedtf-neutralize-xpatch
\newif\iflistingsloaded%
\IfPackageLoadedTF{listings}{%
  \listingsloadedtrue%
}{%
  \listingsloadedfalse%
}

\iflistingsloaded%
  \renewcommand{\lstlistoflistings}{\begingroup
  %\tocsection
  %\tocchapter
  \tocfile{List of \lstlistingname{}s}{lol}
  \endgroup}

  % this is a workaround for including listings with latexmk..
  % This can be fixed
  % - as shown below
  % see https://tex.stackexchange.com/questions/685257/
  % latexmkcan-include-files-created-during-the-latexmk-run-except-with-lstinp
  % - 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
\fi
```

Listing 3.3: A patch of the `listings` package

```
\IfPackageLoadedTF{luamplib}{%
  \newcommand*\inputmpcode[1]{\begin{mplibcode}input #1\end{mplibcode}}
}{}% \IfPackageLoadedTF{luamplib}
```

Listing 3.4: A patch of the `luamplib` package

invocation of `\DocumentMetadata` corrupts reproducibility of the created PDFs. After this is fixed, the `\setMinorVersionPdf` shall be removed again.

Another class of commands provided is represented by `\textttNoChk` which sets the argument in typewriter font just like `\texttt` but for which checks by `chktex` are suppressed. Another example for this kind of command is `\inputNoChk` which may be used to input either generated material like TikZ, or text which is no LaTeX at all. For details see Sections 3.5.1 and 3.6.3.

As the configuration files described above, `header.tex` is intended to be injected in phase `validate`.

Note that `header.tex` is written for use of

- various LaTeX enginess, `lualatex`, `xelatex` and `pdflatex`
- various document classes, specifically `article`, `book`, `beamer` for presentations, `leaflet` and the letter class `scrlttr2`.
- for creating PDF files, but futher formats created with `tex4ht` as well
- direct creation of PDF and via intermediate DVI/XDV

Which packages are loaded at all and if loaded their options, depend on the LaTeX compiler, the output/intermediate format, the document class, packages loaded before and maybe on other criteria.

This is realized with a bunch of if-constructs. In the long run, `header.tex` could be adapted to the configuration as `.latexmkrc`, but currently it detects the use case as the LaTeX engine or the target format and loads the according packages. It is also conceivable to create different headers, one for each document class.

3.5.3 A header file for graphics via package `graphicx`

Chapter 4 lists various techniques to include graphics into a LaTeX document. Most are based on the package `graphicx` and related packages. The injection `headerGrp` is tied to the file headerGrp.tex, which is intended to be included in the LaTeX main file after `header.tex` described in Section 3.5.2 and which loads the packages required for that kind of graphics by need and with the appropriate options, depending on the LaTeX compiler, the output/intermediate format, packages loaded before and maybe on other criteria.

3.5.4 A header file to suppress meta-info for PDF files

Whereas the header file described in Section 3.5.2 is intended to be used in merely any LaTeX main file, the one described here, is optional.

It refers to created PDF files only and does not influence the optical appearance but suppresses writing certain meta-data. The main motivation is security, i.e. privacy, but it can also be used to turn the resulting PDF reproducible.

The injection `headerSuppressMetaPDF` is tied to headerSuppressMetaPDF.tex. Above all, it suppresses information on creation and modification time, on the tool chain used and the trailer identifier. By intention the latter changes in each build run even if the sources are the same. Typically, this is implemented merging the current time into the build process. The trailer identifier is fixed by the header file and so created PDF files created from the same sources are the same, except if date and time are included manually, as e.g. by the command `\today`, except for `xelatex`, which uses the system time to create further hash codes. So, including `headerSuppressMetaPDF.tex` may serve to create reproducible PDF files. As described in Section 5.8, the mainstream technique to reach reproducibility is via manipulating the system time, but if an environment does not support this, including `headerSuppressMetaPDF.tex` is a fallback strategy, if not using `xelatex`.

The extent to which meta info is suppressed is inspired by reproducibility but above all, it is subjective. It is planned to make it configurable, i.e. the file `headerSuppressMetaPDF.tex` is created according to security settings of this maven plugin.

For further information on meta info in PDF files related with security and reproducibility see [Rei23b], Section 4 and how this software treats the handles the issues see Section 5.8.

3.5.5 An installation script for VS Code Extensions

Calling from project root

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

injects the according files instVScode4tex.sh and `.latexmkrc`.

If the editor VS Code is already installed, the script `instVScode4tex.sh`, installs and updates all extensions of VS Code the author used to write LaTeX-code. Project `https://github.com/Reissner/QMngMnt` uses the script for automation of installation and update. It is the only injected file which is executable.

Pasting .latexmkrc, which is just Perl code, into VS Code, one can see the highlighting, of course provided the extensions given by `instVScode4tex.sh` are installed; The configuration file `.latexmkrc` for the development tool `latexmk` is in fact a Perl script.

3.5.6 Scripts in conjunction with reproducibility

Calling from project root

```
mvn latex:inj -Dlatex.injections=ntlatex,vmdiff
```

injects the according files `ntlatex` and `vmdiff` in the root directory.

The injection `ntlatex` injects the file ntlatex which runs the LaTeX compiler specified in the pom, or in the magic comments if present, to create a PDF file. As usual, magic comments override configuration in the pom. Also, it takes into account whether the PDF file is created via intermediate DVI/XDV files or not, depending on the configuration.

This invocation takes also processing time and the timezone into account to guarantee reproducibility if so configured. As `latexmk` is, also `ntlatex` shall be independent of the configuration given by the pom. This is realized in the same way, namely by encoding the configuration in the injection `.latexmkrc`. The downside is, that `ntlatex` like `latexmk` requires Perl to work. For details see Section 5.8 on reproducibility.

But if `ntlatex` so close to `latexmk`, why is it needed in addition? It is because `latexmk` won't recompile, if the expected PDF file exist already and no sources changed. So `ntlatex` is needed to force recompilation.

Complementary to this `vmdiff` is a diff tool for PDF files combining visual equality checked with `diff-pdf-visually` with equality of metadata checked via `pdfinfo` if the files are visually the same. It is realized as a bash script vmdiff and requires no installation except `diff-pdf-visually` and `pdfinfo`.

3.5.7 Script `(de)pythontexW` patching `(de)pythontex`

Calling from project root

```
mvn latex:inj -Dlatex.injections=pythontexW,depythontexW
```

injects the according files pythontexW and depythontexW which just invokes `(de)pythontex` but does not simply output feedback on `stdout` but besides doing so writes it in a log file. This is needed to provide an interface usual in the TeX ecosystem.

Note that all this is specific for unix -like operating systems but can be easily adapted to windows.

3.6 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 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, IllO ("input looks like 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 when 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 LaTeX engine 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[2]. 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 (it may or may not be the

[2]What is worse, there are cases where the console output fails to contain a hint to rerun some program.

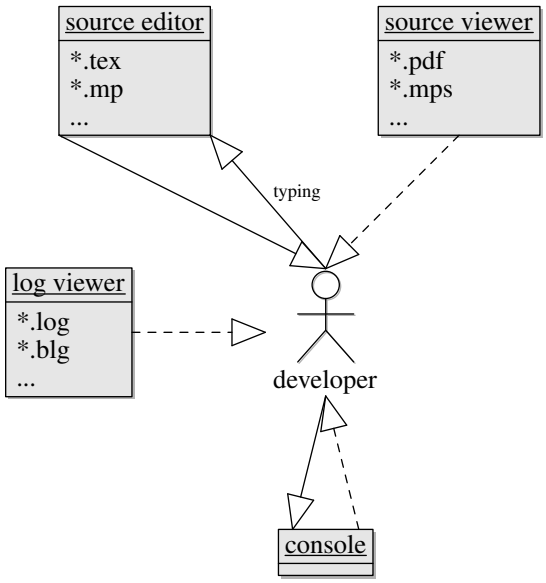

Figure 3.1: Document development with base tools

author) 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. If this data comprises commands the lines are solid, else they are dashed.

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

At first sight, 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. Since the LaTeX-builder is also a checker tool, supervising even warnings, e.g. on bad boxes, and by default invoking `chktex` and monitoring its log file, and since compilation may always fail, the LaTeX-builder may initiate 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 detail, in particular the other tools supporting document development and their interaction. With this background in mind, it is easier 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 3.6.1. 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 3.6.2.

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 3.6.3.

The goals `grp` and `clr` described in Sections 3.6.4 and 3.6.5 make sense only in the context of document development. For details see these sections.

Finally, Section 3.6.6 is on installing extensions for document development on the editor VS Code. To that end, this software provides an installation script.

3.6.1 Editors, viewers and LaTeX

Although there are alternatives like Emacs with extension AUCTeX, the author recommends using VS Code in conjunction with extensions to write and build LaTeX documents and to view the results on `okular`. The recommended extensions are those installed by the script `instVScode4tex.sh` described in Section 3.6.6.

Most of the recommended extensions of VS Code are to highlight the code of the various file types, one, LTeX is a spell and grammar checker, but the central extension is `james-yu.latex-workshop` which also provides build functionality. Among the build "recipies" is `latexmk (latexmkrc)` which is recommended because it integrates well with build tool `latexmk` described in Section 3.6.2 in a way which integrates `latexmk` well with this LaTeX builder. Note that LaTeX Workshop also offers a command "clean up", corresponding with goal `clr` of this software which is described in more detail in Section 3.6.5

As a PDF viewer, we use `okular` with settings given by the menu "settings" and submenu "configure okular". To make `okular` update as soon as the PDF changes, in tab `General`

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

To enable backward search described below, in tab `Editor` choose "custom editor" and type

```
code -r --goto %f:%l
```

Together the "general" settings make `okular` update automatically when a new PDF occurs.

As an HTML viewer any of the usual browsers is usable; they all update as soon as the rendered HMTL file changes.

Still it is a problem to synchronize editor and viewer. As far as the author knows, synchronization is possible only for PDF viewers. Synchronization means at least that for a position on the editor, the according position on the viewer must easily be found and vice versa. Even better would be if moving in the editor selects the according site at the viewer and the other way round. These two features are called *forward search* and *backward search*, respectively. If the LaTeX main file has a setting `synchtex=1` or `synchtex=-1`, then the created PDF has the according information. Then for VS Code with LaTeX Workshop offers forward search: the keystroke `ctrl-alt-j` makes the viewer move to the site corresponding with the cursor position. For the viewer `okular`, backward search is configured in tab "editor" as described above, and it works with the browse tool just hovering over the location of interest and pressing shift plus mouse left key.

This software supports forward and backward search in that it offers injection of a header file `header.tex` which sets `synchtex=1` and offers injection of an installation script `instVScode4tex.sh` which allows installation of the relevant extensions of VS Code.

Besides the separation of editor and viewer, the time delay between writing and reading disturbs document development. LaTeX has a way to speed up compilation: compiling only parts of a longer document which are under construction and which may thus change. These parts must be in separate TEX files and must be included with `\include` not just input using `\input`. With the command `\includeonly` one can specify the files to be recompiled. This works particularly well for document class `book` when including chapters because each chapter starts with a new page, so, page breaks are the same whether compiling a chapter with `\includeonly` or compiling the whole document.

This plugin supports partial compilation insofar as the goal `clr` described in Section 3.6.5 eliminates additional AUX files tied to included sections.

3.6.2 The build tool `latexmk`

Essentially, it is possible to compile latex files only with editor, viewer and a console. Let us collect the challenges. The document may contain graphic files which must be precompiled by further tools which must be invoked a priori on the console. For FIG files this is `fig2dev`. This invocation must be repeated as soon as a FIG file changes[3].

Then a LaTeX engine like `lualatex` must be invoked. Typically, the LaTeX engine must be invoked more than once and besides the LaTeX engine some further auxiliary programs like `makeindex` must be run. The console displays indications

[3]Typically, this triggers a sequence of invocations of converters along files one depending on the other.

to the user what action to be taken next. Normally after invocation of an auxiliary program, the LaTeX engine must be rerun at least once. Each of the programs may fail. Most of the programs write success messages and more detailed information containing error messages, warnings or just information messages on the console and in their respective log files. Potentially, these influence the actions the user must take next.

What is needed, is a tool for orchestration of the basic tools: Orchestration means invoking more basic tools in a reasonable order and supervising the results, at least success and to react appropriately. This frees the user from deciding which of the many auxiliary programs are to be invoked next and whether the LaTeX engine is to be invoked once more to get final correct output. Also, an orchestration tool detects if a build fails or ideally even if a warning indicates that the result is not correct or maybe only not ideal.

There is a tool doing this work, `latexmk`, except that it does not care about warnings.

If something goes wrong, and it is not clear what, it is typically a good idea to fall back to the more basic tools. A great point with `latexmk` is, that this is possible without any problem, and it is as simple to switch back from basic tools to `latexmk`. This is what we mean saying that `latexmk` *integrates* the basic tools.

The best way to invoke `latexmk` for document development is

```
latexmk -pvc latexFile
```

According to [Col23], Section "DESCRIPTION", the option `-pvc` is shorthand for "preview continuously", a kind of nonstop mode: The PDF file, or whatsoever is created, then a viewer is opened in the background if not yet open and then `latexmk` monitors changes of dependencies and triggers a rebuild each time a change is detected performing a proper sequence of invocations of LaTeX engines and auxiliary tools. Note that `latexmk` does not stop after finishing a compilation, whether successful or not. Instead, it awaits a change of a source file which triggers a new run of some basic tool until interrupted by the user. The option `-pvc` is described in more detail in [Col23], Section "LATEXMK OPTIONS AND ARGUMENTS ON COMMAND LINE". One detail to be added, mentioned in [Col23], Section "DESCRIPTION", is, that `latexmk` detects dependencies based on the FLS file written by the LaTeX engine when invoked with the `-recorder` option.

A small fallback step advisable if something goes wrong is to interrupt continuous viewing and to invoke `latexmk` without options. Then `latexmk` performs a single build and finishes; no viewer is opened. This may help in understanding the problem, but in general, it is advisable to go back to basic tools like `lualatex`. To understand the build process from scratch, erase all created files by `latexmk -C` or all intermediate files by `latexmk -c`, which does not erase the resulting PDF file, before using the basic tools.

Differences of `latexmk` with this LaTeX builder

Let us discuss the differences between `latexmk` and this latex plugin: First, the plugin runs within a maven process which introduces a lot of overhead. So this cannot be as fast as `latexmk` is. In addition, a maven plugin cannot open a viewer. Moreover, the plugin is designed to build all LaTeX main files and not to focus on a single one. In many cases, more than one output format shall be created. The latter properties which are disadvantages in the context of document development, can be overcome, by specifying a single target in the setting `targets` or by invoking goals with a single target, e.g. by `mvn latex:pdf` and to restrict to building a subset of files and if needed a single LaTeX main file with the settings `mainFilesExcluded` or `mainFilesIncluded` described in Table 6.1 on page 122.

Another difference is, that by default, the plugin cleans up the folder with the TEX sources, and only the resulting file, e.g. PDF is copied to the target folder before cleanup. To be more precise, only the files present before the build are kept, possibly updated, all the others are removed. This is appropriate for a maven plugin but destroys log files containing vital information if the build goes wrong. Still if a file is interesting it may be created by touch or by some basic latex tool as `lualatex` or `makeindex` and then a build done by this plugin will pertain the file updated by the build process. For document development, the parameter `cleanUp`, also described in Table 6.1, which is `true` by default, can be set to `false` so that no file in the latex directory is deleted.

So, it is clear that this plugin is for final global build with a lot of supervision sensitive to detecting caveats. To overcome these, further development of the document is necessary, which is better done individually on the problematic document with `latexmk`. In a sense `latexmk` is the fallback to this maven plugin as much as `lualatex` is the fallback to `latexmk`.

To make this work, this plugin must integrate `latexmk` as `latexmk` integrates `lualatex`. This is guaranteed, if this plugin can write a config file `.latexmkrc` which causes `latexmk` to behave like this plugin. This is exactly what injection of `.latexmkrc` is intended to do according to Section 3.5.1. Note that this feature is just offered, but the user may also use his/her own file `.latexmkrc`.

Based on injection of `.latexmkrc`, this plugin may even use `latexmk` as a means to build bypassing its internal build rules. For motivation of this feature and for details in implementation see Section 5.9.

There is a difference in the build processes (except if this plugin uses `latexmk`) concerning mostly graphic files: `latexmk` detects dependencies via the `-recorder` option of the latex generator and creates or recreated what is new or changed. This is more elegant than the idea of this plugin which is creates a fixed set of graphic files first and is from that point on based on detecting hard coded set of files and tracing log files. In other words, `latexmk` has no graphic preprocessing as Chapter 4

describes for this build tool. This offers the advantage, that `latexmk` never creates graphic files which are later not needed for inclusion. Nevertheless, to deal with graphic files which are to be created in the course of the build, `latexmk` runs the LaTeX engine in `nonstopmode` mode. Still, the run of the engine is interrupted, a single graphic file is created according to some rule and then the engine is rerun. For a document with 10 graphic files to be created in the course of the build, for `latexmk` only the 10th runs of the LaTeX engine is completed. In contrast, this plugin requires a single run, so performance is significantly better. The use of `\IfFileExists` is not really elegant but prevents `latexmk` from frequent reruns and in some cases is a technique to make the build process with `latexmk` work. One of these cases, related with using `listings` is discussed below in this section.

More general, there are cases, where this latex builder succeeds but `latexmk` does not. As this latex builder may invoke `latexmk` either in general or for selected files, this latex builder is mightier than `latexmk`. If both approaches succeed, the results shall be the same for this plugin and for `latexmk`.

It is possible to combine this plugin with `latexmk` to speed up `latexmk`:

```
mvn clean validate latex:grp
```

cleans like `latexmk -C` and in `validate` invokes goal `inj` injecting `.latexmkrc` to configure `latexmk` and maybe `header.tex` necessary to compile the latex files at all. Finally, goal `grp` creates the graphic files which speeds up `latexmk`. Of course the above maven invocation is also a good initialization for building with the basic tools without `latexmk`.

Finally, goal `clr` tied to phase `clean` erases all intermediate files and thus makes the next build independent of the previous one.

For further reading on goal `grp` creating graphics files see Section 3.6.4, Section 3.5 is on file injection and goal `clr` to clear created files is described in Section 3.6.5.

How `latexmk` is integrated

Finally, we show how this plugin may support `latexmk` where. To understand in which sense, one must dive very deep. In short, injection of a header patches package `listings` in a way that saves performance of `latexmk`. Let us elaborate.

This software and `latexmk` follow a different philosophy in finding dependencies: Whereas this software creates image files in advance before invoking a LaTeX engine, `latexmk` first calls the LaTeX engine in `nonstopmode` to avoid a stop because of a missing file. Then the file is created using the appropriate rule (hopefully unique) and the emgome is run again, this time passing the inclusion of the first created files failing at the next one. To find out that another rule is needed, `latexmk` parses

the LOG file of the latex compiler. As the packages write log messages in their own style, this is the point where the solution is no longer generic and so it is no wonder that there is at least one kind of inclusion which does not work that way: inclusion with `\lstinputlisting` provided by `listings`. In fact, the author has 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 for the second problem is by patching `listings` as described in Section 3.5.2.

The suggested workaround for the first problem is creating graphic files using goal `grp` as described in Section 3.6.4 before invoking `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 engine is required: The initial run is interrupted with the first missing file. Then that file is created by an appropriate rule and the LaTeX engine 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.

3.6.3 Checks in the context of document development

The target `chk` just invoke the tool `chktex` and logs finding in a CLG file. It is invoked as the final quality check for the documents created from latex sources. But if this check fails, there is a transition to document development. As said in Section 3.6.2 on running this plugin on a single file with a single target applies here also. But here again, this plugin is not the first choice: Better is to invoke `chktex` directly and to eliminate the warnings iteratively. Since the file `.chktexrc` injected by this plugin as described in Section 3.5 configures `chktex` whether `chktex` is invoked directly by the user or via the plugin in goal `chk`, the results are the same. In the wording coined in Section 3.6.2, this plugin integrates `chktex` very much the same way as it integrates `latexmk` namely by injection of a config file.

The config file `.chktexrc` in turn is adapted to the header `header.tex` which is also injected. In general, `.chktexrc` excludes content of environments and of arguments of commands defined in packages loaded by `header.tex` or defined therein directly. A nice example of another kind of synergy is the command `\textttNoChk` defined in `header.tex`. Functionally, it is just `\texttt` which sets the argument in typewriter font, but in `.chktexrc` it is listed among the commands the arguments of which shall not be checked by `chktex`.

After eliminating warnings until direct invocation of `chktex` displays no warnings, one can be sure that also check with goal `chk` of this plugin does not yield

warnings.

3.6.4 Goal Graphics `grp`

In the context of document development, typically compilation is done by basic tools like `lualatex` or by an orchestration tool like `latexmk`. Nevertheless, since separation of builds is desirable, intermediate files like graphic files are not present. Maybe they are removed by cleaning.

The

TBD: check whether this is really needed: is also described in section on latexmk. Maybe we need a section on this plugin describing grp and clr uniformly. Maybe also first write on chktex and its relation to this plugin.

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 engine like `lualatex` and also to compile smoothly with `latexmk`.

3.6.5 Goal Clear `clr`

When invoking this plugin as a final build, `cleanUp` is set to its default `true`. Thus, all files not present at the beginning of the build process are removed. As a consequence, there is no need for a separate goal `clr`. This comes into the game only in the context of document development. Either `cleanUp` was set to `false` or other more basic tools created intermediate files which must be deleted by `clr`.

Cleaning is vital because it makes the next build independent of the previous one. Deletion is driven by a regular expression `patternCreatedFromLatexMain` described in Table 6.1 on page 122. Completeness can be guaranteed only if the set of loaded packages is limited. Of course, only created files shall be deleted. For packages introduced in the injected header `header.tex` described in Section 3.5.2, this shall be the case. The author's criterion for a correct regular expression is, that after deletion exactly the files under version control remain.

The goal `clr` corresponds with `latexmk -C` and is tied to phase `clean`.

Clearing comprises files created by the goal `grp` and by any other goals. Note that AUX files are deleted if they belong to a LaTeX main file or to an included file.

The most interesting files are those created by injection, i.e. by goal `inj` like `.latexmkrc`: As pointed out in Section 3.5, each of the files in question is deleted only 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.

3.6.6 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.

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 are those described

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.

3.6.7 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 engine, 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 engine 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 3.5. 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.

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 instVScode4tex.sh 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.

3.7 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, both goals are 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. Also Listing 2.4 shows a recommended configuration for the goal `latex:inj` which determines injected the files.

3.8 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>2.1</version>

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

Listing 3.5: Configuration without cleanup

Chapter 4

Graphics and 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 a LaTeX document, but all need loading of according packages. For an overview of the graphic formats and the packages needed for their support see Section 4.1. The set of available graphic formats is extended by *preprocessing*, i.e. by processing prior to the LaTeX engine. Preprocessing mainly consists in converting graphic formats not supported by LaTeX packages into graphic formats supported by some LaTeX packages. Section 4.2 provide vital information on the target formats.

This software uses preprocessing for graphics only. Note that preprocessing is a design decision on the build tool and e.g. `latexmk` has no preprocessing at all. For details see Section 3.6.2.

Table 4.1 gives an overview over the formats supported via preprocessing. The first column lists the formats, the second one at least one editor for the format, and the last row contains the parameter to configure the preprocessing tool and give the default tool as an example. Sections 4.3, 4.4, 4.5, 4.6 and 4.7 treat each format separately. For all but PNG and JPG considered in Section 4.7, preprocessing is just conversion of the format into another format directly supported as described in Section 4.1. Historically the latter two required preprocessing to determine the bounding box was needed. We still support this to support historical techniques and to be sure to be able to reconstruct historical documents. Support for further formats can be easily added. If there is some need, please write an email to the author.

Of course, to support a format, the preprocessing tools must be installed. It is advisable to have also an editor installed. Sometimes the editor is used also as converter as for `inkscape`. For human-readable formats like `fig`, it often makes sense, to use both the graphical editor and the textual one. Note that `vscode` supports the given formats more properly, if the extensions described in Section 3.5.5

are installed also.

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

Table 4.1: Overview over the graphic formats supported via preprocessing

4.1 Graphic formats and packages supporting them

Find below a list of packages either allowing to include directly certain graphic formats, or helping with graphics indirectly. Although strictly speaking these techniques do not need special treatment of a build tool, this software supports these techniques by providing header files by injection loading the needed packages.

We also describe in which sense these packages support graphical preprocessing.

`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]. Note that PDF and EPS are target formats for graphical preprocessing, where PDF is embedded into PDF and EPS is embedded into DVI/XDV. As described in Section 4.5, also MPS, the target format for metapost is included using `graphicx`.

`transparent` allows specifying transparency in graphics. 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 the LaTeX engine must get along with it. Note that this applies only for output format PDF and in particular not for `xelatex`. For details see [Obe16b].

`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 [Tan23] is just in LaTeX format. Thus, it can be included directly and does not require any preprocessing. Still what is needed is a good graphical editor like `tikzedt` with online manual [TW12]. In later

versions of this software, 3.x or so, it is planned that TikZ is used as new target format for graphical preprocessing, replacing the current combination of LaTeX for texts and PDF/EPS for proper graphic.

`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.

`xcolor` allows using colors in graphics. Even if the author does not use colors in graphics, several formats, like FIG, GP (GnuPlot file format) 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`. In this sense its role is comparable to that of `transparent`. On the other hand, the use of `xcolor` is not specific to graphics. For details see [Ker16].

`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 engine is required also, but it is not preprocessing.

Section 3.5 is on injection of files and in particular header files:

`header.tex` treated in Section 3.5.2, is a general header file intended to be included into all LaTeX files. Since the packages `import` and `xcolor` are generally useful, not only in the context of graphics, they are among those loaded in `header.tex`.

`headerGrp.tex` described in Section 3.5.3 in contrast, is a header file loading graphic specific packages related with `graphicx`, loading also `transparent` and `bmpsize`.

The header files adapt the loading of the packages to the context, in particular to the target format. Note that `headerGrp.tex` must follow `header.tex`.

The package `tikz`, although a pure graphic package is very specific and not related to `graphicx`. Thus, it must be loaded separately. The same holds for `pythontex`.

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 4.3 shows how `fig2dev` converts fig-files into LaTeX-files containing text and including graphics in as PDF files. Likewise, Section 4.4 describes how gnuplot converts gnuplot-files into PDF files. An interesting alternative to gnuplot for computing pictures is MetaPost described in Section 4.5. A more elaborate alternative to fig-pictures are SVG pictures described in Section 4.6 Also several formats collected in Section 4.7 may be included as is.

4.2 Target formats for preprocessing

At a first sight, PDF seems the ideal target format for any kind of preprocessing: It is really mighty enough to display pictures in any source format without loss of information and even without change in appearance, and for modern LaTeX implementations directly creating PDF files, the LaTeX-package `graphicx` allows including graphics as PDF files in LaTeX-files.

At a second sight, the source formats under consideration offer pictures mixing vector graphics and texts and in particular formulae set in LaTeX style. Preprocessing is based on on-the-shelf converters and if targeting PDF, the texts originally in LaTeX style change their appearance. To keep up LaTeX style, they provide mixed export consisting of a PDF file containing proper graphics without texts and a TEX file containing the texts in proper location and an `\includegraphics` command including the created PDF file. This mixed conversion is used for all kind of preprocessing.

Note that we could have used the ending TEX for the texts, but we opted for a specific ending PTX (pdf/postscript TEX format; home-brewed) signifying that the file is created and thus does not slow down search of LaTeX main files.

But still there is another problem with PDF as target format: Traditionally LaTeX produced output in the DVI/XDV (eXtended Device Independent; an extension of the traditional output format DVI of LaTeX engines, today widely replaced by PDF)-format which is still used to create HTML-output. For LaTeX engines `pdflatex` and `lualatex`, DVI output is specified with option `--output-`

`format=dvi`. It turns out, that with this setting, PDF files cannot be incorporated with `\includegraphics` command. Instead, one must use EPS files. Fortunately, the graphic converters used also support combined TEX/PTX and EPS formats. We ensured that `\includegraphics` in the PTX file specifies the file without ending so that the PTX file is the same, whether it encloses a PDF file or an EPS file, and we provide both, a PDF file and an EPS file[1]. That way, both, `pdflatex` and `lualatex` choose the EPS file or the PDF file depending on whether the output format is `--output-format=dvi` or `--output-format=pdf` which is the default. Note that `xelatex`, which always creates an intermediate XDV file (which is a special kind of DVI file), acts differently: If present, it prefers including the PDF file, if absent, but there is an EPS file instead, it uses this without making any difference.

Although this is beyond necessity, let me state that `pdflatex` and `lualatex`, while not accepting inclusion of PDF files in DVI mode, EPS files are accepted in PDF mode for more modern versions of the LaTeX engines, but this leads to creation of intermediate files `xxx-eps-converted-to.pdf`, which are not cleaned up in target `clr`.

Whereas PDF and EPS files both are offered, only one of them is included for a specific configuration. This is in contrast to other formats described in Section 4.7.

Although PTX is just a TEX format, it is special in that it presupposes that some packages are loaded before being included. The packages which are not specific for graphics like `xcolor` are loaded in `header.tex` described in Section 3.5.2, whereas the ones specific for graphics, above all `graphicx`, are loaded in `headerGrp.tex` as described in Section 3.5.3. The packages actually to be loaded and their respective options depend on the configuration.

Note that PDF and EPS file may be created by preprocessing but also as proper sources not created at all, even in a single document. Goal `clr` deletes the according files `xxx.pdf` or `xxx.eps` only, if an according source like `xxx.fig` exists. Else it is treated as proper source and is not deleted.

In the future, the combination of PDF/EPS and PTX files may be replaced, at least partially, or supplemented by TikZ files. It turned out, that the converters under consideration support more and more conversion into the TikZ format which can represent both, proper vector graphic and also LaTeX texts like formulae. Using TikZ as intermediate format has the advantage, that the working space is polluted less with generated files, preprocessing is speeded up because fewer files are created and in some cases, less processing steps are needed. Another advantage is, that the internal dependency recording of LaTeX engines made available through the FLS (FiLeS dependencies: list of files the according tex file depends on; output

[1] Of course, here a more sophisticated technique is conceivable, recognizing the required format and generating the specific one if missing.

format of LaTeX engines if used with option `-recorder`) file is accessible. As in the current technique using PTX files instead of TEX files, we could put the TikZ into TEX files, but we opt against it for the same reasons.

Note that PS is not supported because it misses the bounding box. If adding it, one arrives at the EPS format.

4.3 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 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 4.1.

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.

```
\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 4.1: The ptx-file for a fig-file

- 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`. 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.

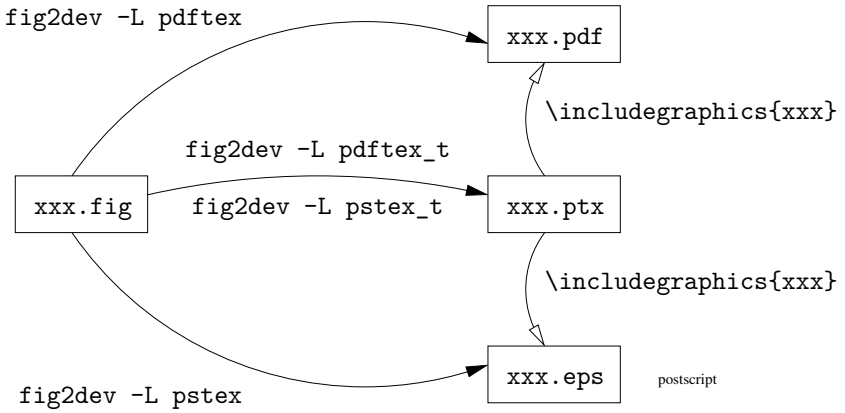

Figure 4.1: Conversion of a fig-file into pdf-, eps- and ptx-files with inclusions

4.4 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 [WK23].

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 handwritten 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 VS Code, one can see the highlighting, of course provided the extensions described in Section 3.5.5 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 [WK23], Part IV, `gnuplot` supports output formats through so-called *terminals*. Among those are several ones intended for inclusion into LaTeX-files, like `Cairolatex`, `Epscairo`, `Epslatex`, `Latex`, `Lua (tikz)`, `Postscript`, `Ps(la)tex`, `Pstricks`, `Texdraw` and `Tikz` which is in fact equivalent with `Lua (tikz)`. Comparison with the manual [WK16] for older versions of `gnuplot` shows that support of `Eepic`, `Mp` and `Tpic` ended. 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 4.3. Also, `gnuplot` pictures may be exported in MetaPost format which in turn may be included in latex as described in Section 4.5.

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 4.3, 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.

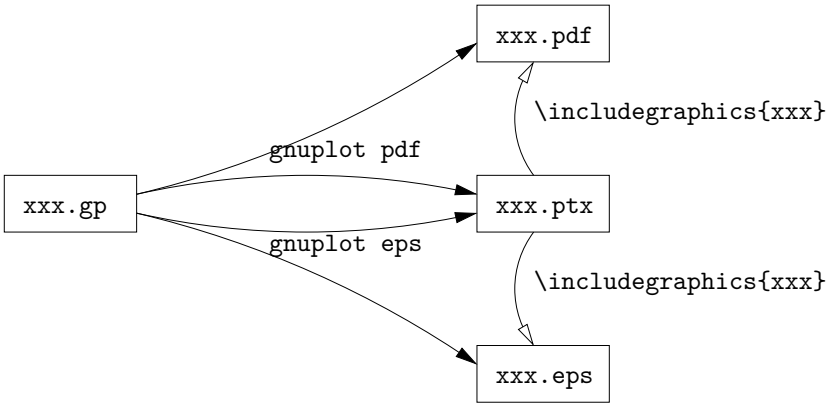

Figure 4.2: Conversion of a gnuplot-file into pdf-, eps- and ptx-files with inclusions

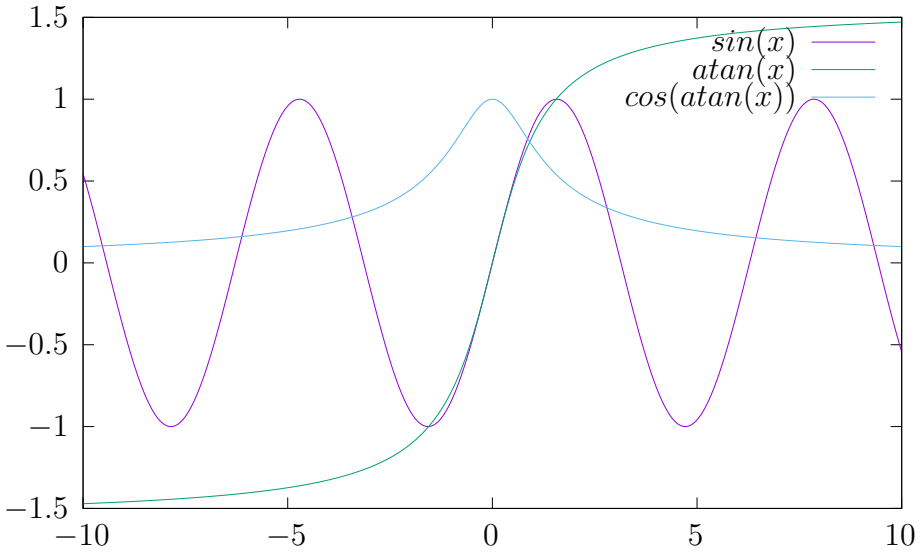

Figure 4.3: Converted sample gnuplot-file into ptx and pdf files

4.5 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 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 [Hob24], 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 4.2 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 described in Section 3.5.5 are installed.

Listing 4.2 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`[2], 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.

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 [Hob24], 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. Caution: in the manual, the variable is referred to as “key”.

The most basic setting is `outputformat:="eps"` which is the only setting appropriate for latex. So don’t change[3]. Note the strange default setting for

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

[3]Note that `metapostCommand` may also besides EPS output SVG and PNG, just by setting

```
%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 4.2: An example file in MetaPost

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
```

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 4.2, line 21 includes a label via a note implicitly, and for the material between `btex` and `etex` uses TEX. The manual [Hob24], 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 [Hob24], 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 4.2, 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:

`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

- 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 MPX (metapost TEX output: texts)-file `xxx.mpx` containing the LaTeX text of the figure; this is not created if there is no such text.

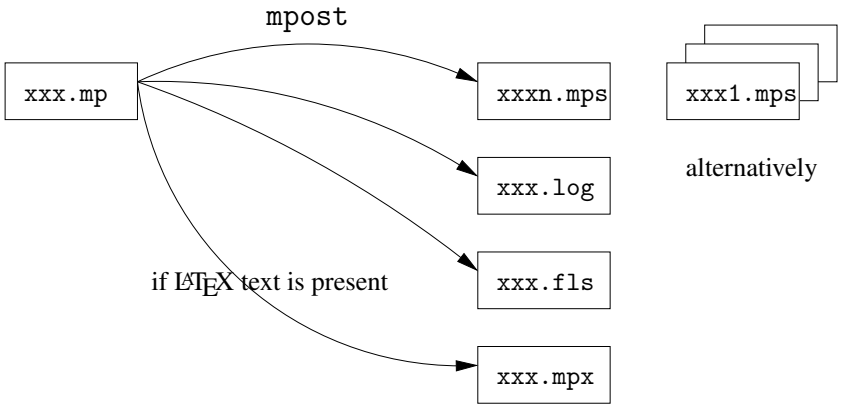

Figure 4.4: Conversion of a MetaPost-file into an mps-file

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

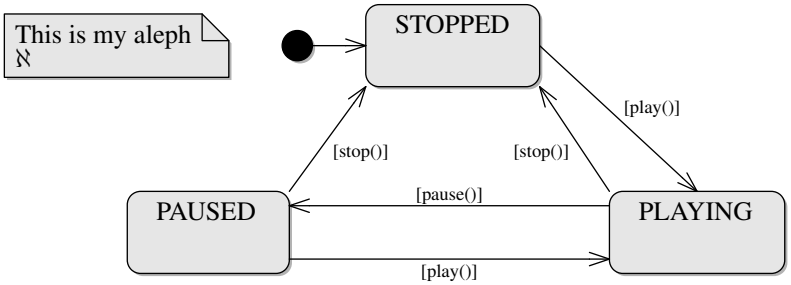

Figure 4.5: Converted sample MetaPost-file included as mps-file

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. As described in Section 3.5.2, this software can inject a header which loads the header and enhances it providing the additional command `\inputmpcode` which allows also load MetaPost from a file. The latter is preferred to direct inclusion with the `mplibcode` 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}
\inputmpcode{F4_05someMetapost}
}% to recover from redefine {manualC4graphics.tex}
```

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

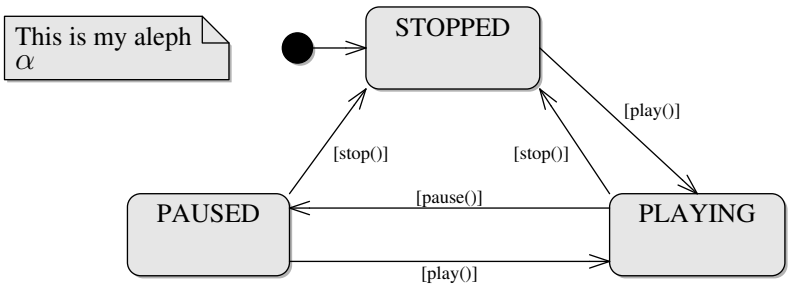

Figure 4.6: Sample MetaPost-file included via `luamplib` for lua(hb)tex

4.6 Inclusion of SVG-files

Comparable with the xfig-format described in Section 4.3 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 4.3 and for gnuplot-files in Section 4.4: 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 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 4.3 and 4.4.

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 the future.

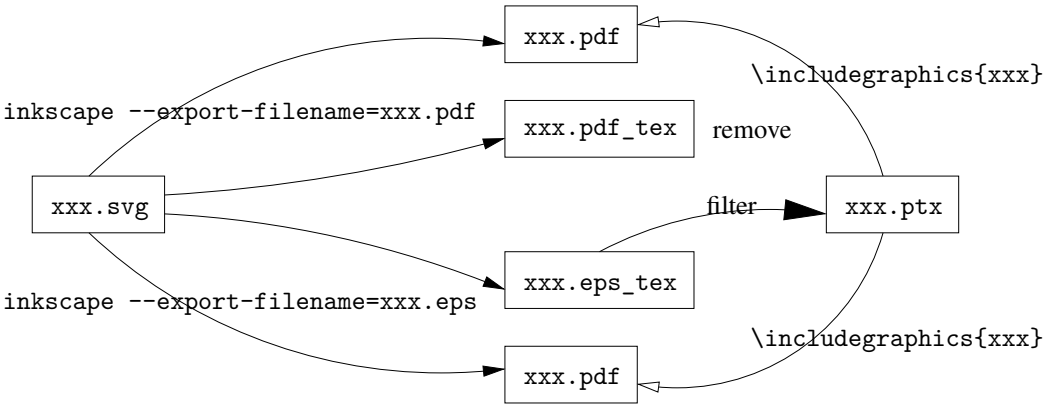

Figure 4.7: Conversion of an SVG-file into pdf-, eps- and ptx-files with inclusions

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 graphical 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 VS Code, one

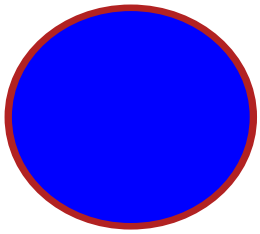

Figure 4.8: Some svg-picture with text FIXME: uniformity

can see the highlighting and a preview, of course provided the extensions described in Section 3.5.5 are installed.

4.7 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 seamlessly 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 safe, 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/XDV output 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` files in the format PDF, PNG, JPG are supported. This list may be incomplete.

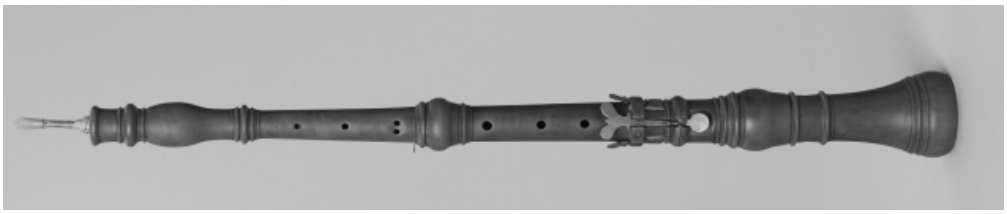

Figure 4.9: Some JPG-picture, directly included

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.

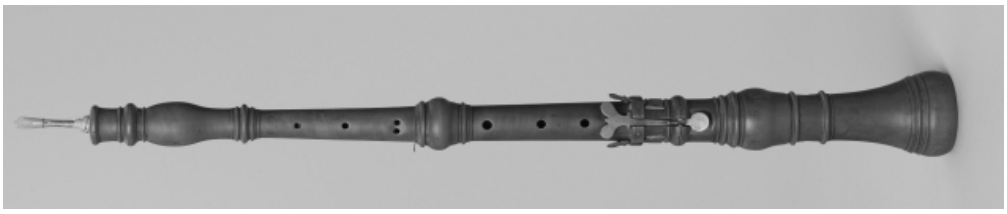

Figure 4.10: Some PNG-picture, directly included

Note that in DVI/XDV mode all usual LaTeX engines 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.

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 describe running and rerunning auxiliary programs like `makeindex` and the LaTeX engine, respectively. The latter may be necessary if certain lists are present like table of contents list of figures or list of tables. Section 5.6 clarifies the exchange of information between the LaTeX engines and auxiliary programs, whereas Section 5.7 essentially describes the exchange of information between individual runs of the LaTeX engine.

Section 5.8 is special in that it is not related with conversion but with checking reproducibility. This LaTeX builder has some built-in build algorithm, but one can also use `latexmk` as a build tool in a way that invokes all tools with parameters given by the configuration. Note that `latexmk` has a different build algorithm, but the results should be the same. This is mainly to integrate document development more seamlessly. For details on motivation and implementation see Section 5.9.

Besides the output formats traditional for LaTeX, PDF and DVI describing e.g. books, Section 5.10 describes creation of HTML, Section 5.11 the creation of ODT and Section 5.12 creation of MS Word formats like DOCX. Finally, also pure text

can be generated as described in Section 5.13.

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 engine `latex2pdf` to a master TEX file `xxx.tex`. The LaTeX engine `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 engine. Besides conversion into PDF format, all engines offer conversion to the older DVI format via option `--output-format` as `lualatex` and `pdflatex`, or the alternative XDV generalizing DVI as `xelatex` does with the option `--no-pdf`.

In fact, the engine `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 engine with package `biblatex` loaded creates a BCF (bibliography content file (?): generated by LaTeX engines if used with package `biblatex`) file read by `biber`. At time of this writing, 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 IND (INDex file containing sorted, unified and formatted index entries, output format of `makeindex` and `xindy`) 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.

A more modern technique to create an index is via `xindy`, but at time of this writing, this software does not support `xindy` yet.

If a glossary is demanded, this can be read off the AUX (auxiliary file: input and output file for LaTeX engines; read also e.g. by `bibtex`) file and a GLO (GLOssary file containing unsorted and multiple glossary entries; output format of LaTeX engines with package `makeglossaries`) file containing the index entries is created and a file with style information. Depending on the configuration, this may be a IST ((make-)Index Style File: output format of LaTeX engines if used with package `glossaries` configured for `makeindex`) file or a XDY (index style file for `xindy`: output format of LaTeX engines if used with package `glossaries` configured for `xindy`) 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 GLS (glossary file containing sorted, unified and formatted glossary entries; output format of the `makeglossaries` tool read by LaTeX engines) file containing the glossary, which is read in the next `latex2pdf` run. For details see Section 5.4.

Besides `makeglossaries`, there is a more modern tool, `bib2gls`, which is not yet supported by this software at time of this writing.

The package `pythontex` allows including python code or related in the TEX (TeX the format, which may also be LaTeX) file and to evaluate it. The first `latex2pdf` run creates a PYTXCODE (Code file consisting mainly of code snippets from the TEX file; output format of LaTeX engines with package `pythontex`) 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 DEPYTXC (File containing information to replace code snippets in the TEX file by the result of their evaluation; output format of LaTeX engines with package `pythontex` if loaded with option `depythontex`) 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 OUT (contains bookmarks: input and output format of LaTeX engines if used with package `hyperref`, file ending seems naive) 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 auxiliary programs mainly to create index or glossaries, but also under certain circumstances bibliographies and inserting invoked code, followed by invocation of the LaTeX engine `latex2pdf` may be necessary.

5.2 Bibliographies

For each occurrence of a command `\cite` in the TEX file, referring to a document with given key, `latex2pdf` writes an according entry `\citation` with that key into an AUX file. Note that, if the LaTeX main file includes other TEX files with `\include`, and the `\cite`-command is invoked in the included TEX file, the `\citation` commands go into the AUX file of that TEX file. Moreover, a `\bibliography`-command in the TEX file writes a link to the BIB files containing

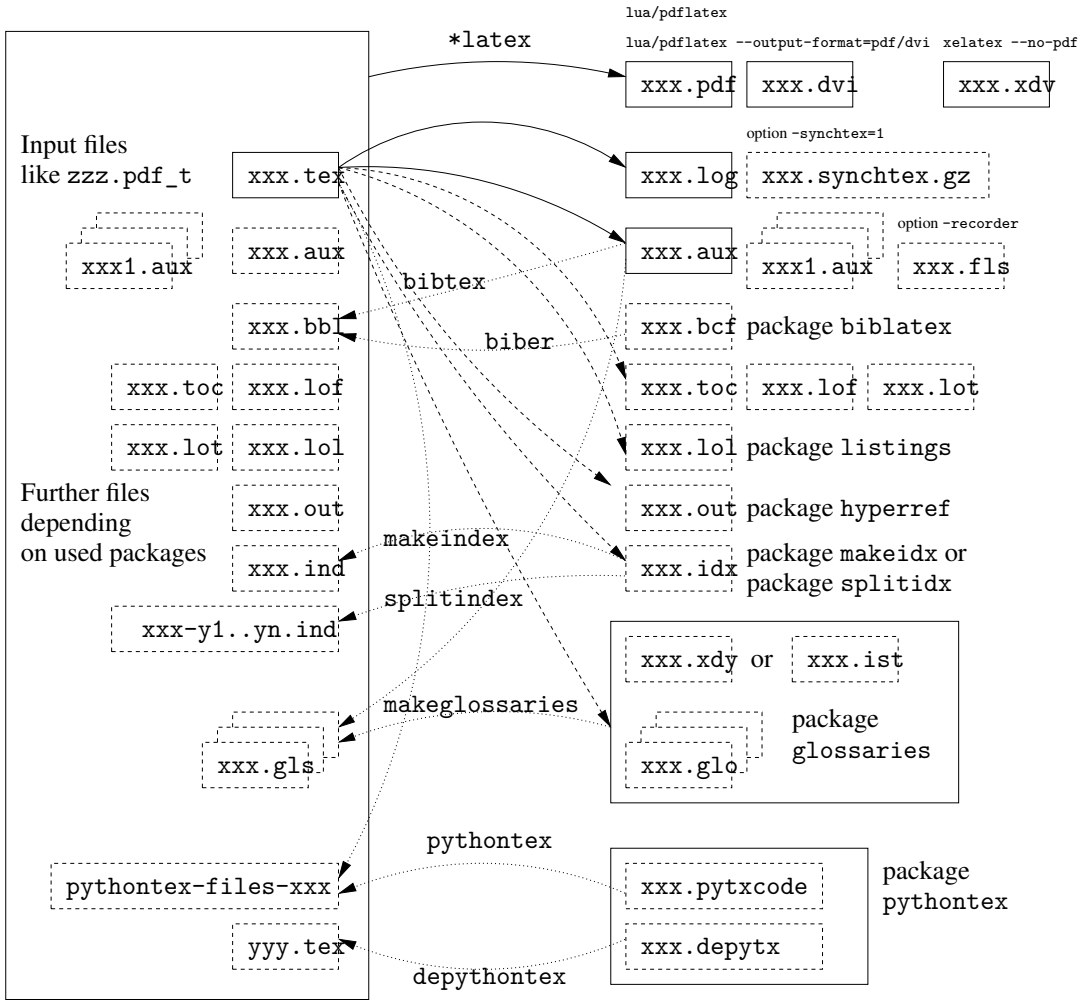

Figure 5.1: Conversion of a TEX file into a PDF, DVI, XDV file

the bibliography data into the (top level) AUX file as `\bibdata`. Note that `\bibliography` accepts a list of BIB files, not only a single one, as maybe suggested by the singular name. The key given by `\cite` commands must refer to exactly one key in the BIB files. Last not least, a `\bibliographystyle`-command in the TEX file writes a link to the bibliography style file which determines the appearance of the bibliography and also the labels and the ordering into the AUX file as `\bibstyle`. Typically, the style file comes from the TEX distribution rather than the user. Its ending is BST (Bibliography Style File read by the `bibtex` tool).

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 also supported like `bibtexu` and `bibtex8` supporting utf8 encoding and others.

Among the tools which are not supported are `biber` and `mlbibtex`.

We run `bibtex` if either command `\bibliography` or `\bibliographystyle` is in the top level AUX file. If there is no `\cite`-command, `bibtex` yields an error. If neither `\bibliography`-command nor `\bibliographystyle`-command are present, then presence of `\cite` yields an error when running the LaTeX engine. So, there is an error if not either all three ingredients are present or neither.

Essentially, `bibtex` extracts the citations in the AUX files, unifies them, i.e. a citation is listed once even if it is used more than once, retrieves the according entries from the BIB files specified, sorts and formats these entries according to the BST file and writes all into a BBL (bibliography for a latex document in latex format: written by the `bibtex` tool and read by LaTeX processors) file which can be included in the next run of `latex2pdf`. Formatting includes associating a label with each key and sorting is based typically on the label. The BBL file consists essentially in a `thebibliography` environment listing the `\bibitem`s. These relate the key and the label given by the BST file and show the text of the bibliography entry.

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 at place of `\bibliography` (which shows why it is singular, although a list of BIB files may serve as source) 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.

This software presupposes, that `bibtex` reads the AUX file and creates a BBL file and also a 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.

Vital information on `bibtex` can be found in [Pat88] and in [Mar09]. Also, [Grä96], Chapter 10 is worth reading in this context.

Note that in the master AUX file one can find also entries `\bibcite` relating the labels for bibliography entries to the representations to be inserted for the `\cite` commands, but it is the LaTeX engine which extracts these mappings from the `\bibitem` entries in the BBL file written by `bibtex`.

The package `tocbibind` described in [WP10], then writes the headline of the index into the table of contents, if the option `numibib` is given.

5.3 Indices

Let us first assume that only a single index is wanted. For each occurrence of a command `\index` or similar (details see below) in the TEX file, referring to an

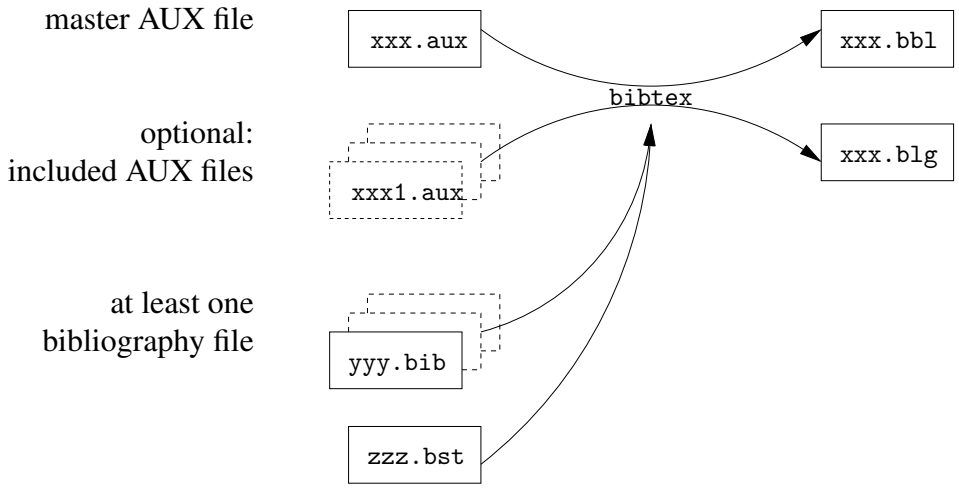

Figure 5.2: Conversion of an AUX file into a BBL file using bibliographies

entry of the index, `latex2pdf` writes an according entry `\indexentry` into the IDX file, provided before the command `\makeindex` was issued.

In case the LaTeX engine 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 in the TEX file specifying an index entry, an `\indexentry` command is written into the IDX file comprising the keyword to be written into the index given by the `\index`-command and the page number where the `\index`-command occurred.

is written to the IDX file as `\indexentry` For example `\index{ant-task}` in occurring on page 3 creates an entry

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

in the IDX file.

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, if the option `numindex` is given.

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.

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.

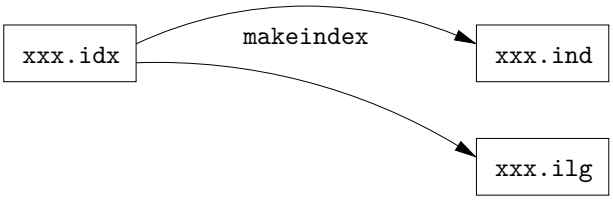

Figure 5.3: Conversion of an IDX file into an ind file

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 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.

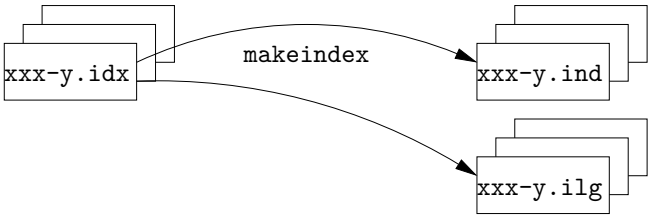

Figure 5.4: Not supported: Conversion of IDX files into ind files

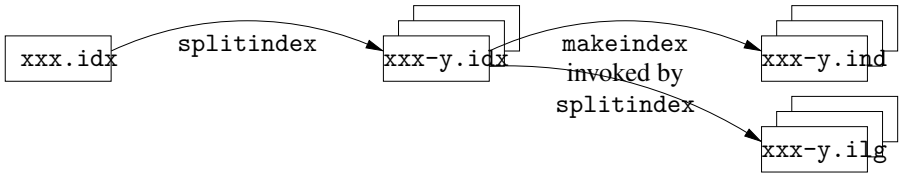

Figure 5.5: Conversion of an IDX file into ind files

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 `--` 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 engine 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 [Tal24b]. By default, package `glossaries` creates a single "main glossary", which can be switched off specifying the option `nomain` described in Section 2.6. In this case at least, more specific glossary types with according headline must be specified. As specified in [Tal24b], Section 2.6, `glossaries` offers `acronyms`, `symbols`, `numbers` and `index`. To avoid collision with indexing as described in Section 5.3, this software does not allow the latter. Moreover, the package `glossaries` even supports user-defined glossary types, but this software does not, mainly to keep the internal build in line with build using `latexmk`. For details see Section 8.4.

Also, the package `glossaries` offers sorting and unifying either via `makeindex` as for indices or via `xindy`, and it offers also to do without external programs. In contrast, this software supports only the variant using `makeindex`.

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 [Tal24b], Section 13 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.

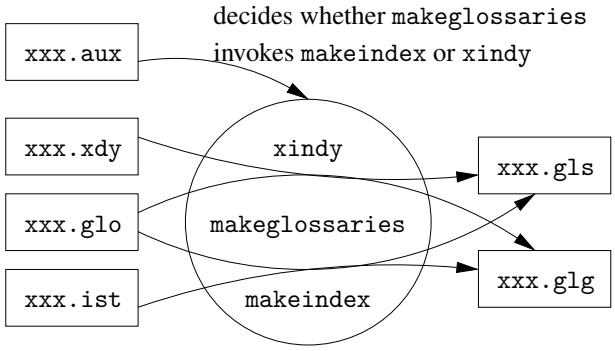

Figure 5.6: Conversion of a glo file into a gls file using `makeglossaries`

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 extend 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 engine 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 in Section 3.5.7, instead of `pythontex` itself. The figure reflects this.

Running the LaTeX engine 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 engine, because files `*.stdout` can impose heavy memory usage and currently `lualatex` is the only engine 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` and place it where it can be found, e.g. in the folder of `pythontex`.

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` and described in Section 3.5.7, 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.

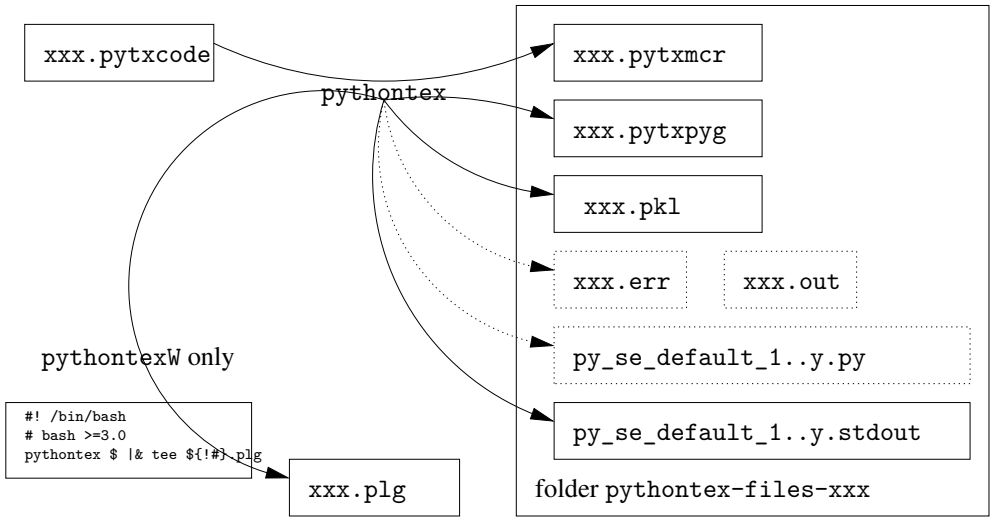

Figure 5.7: Conversion of a `pytxcode` file using `pythontex`

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.

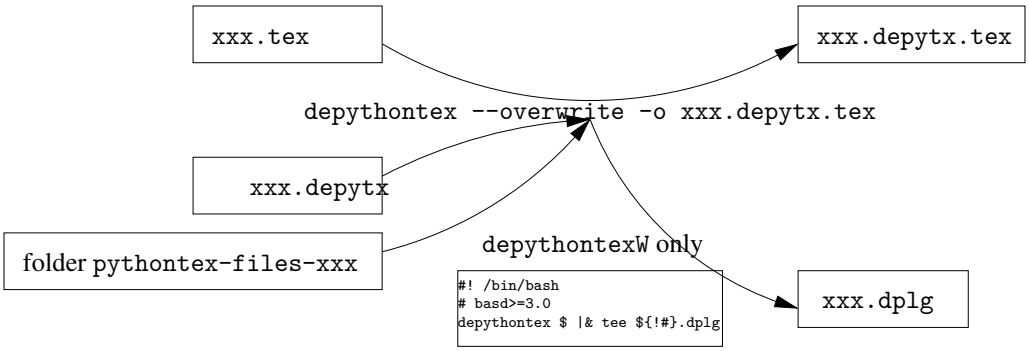

Figure 5.8: Conversion of a `depytx` file using `depythontex`

5.6 Running and rerunning auxiliary programs

After describing the interface between the LaTeX engine and the auxiliary programs in Section 5.6.1, Section 5.6.3 explains why we don't use the package `rerunfilecheck` to determine when to (re-) run auxiliary programs.

5.6.1 The interface between LaTeX and auxiliary programs

Auxiliary programs perform tasks which LaTeX cannot carry out at all or only with bad performance, for example adding bibliographies which comprises sorting or executing program code.

The interface between the LaTeX engine and an auxiliary program is always implemented via files: In the first run, the LaTeX engine writes a file or files specific for the auxiliary program or at least writes entries specific for the auxiliary program in a standard file or even both. Then the auxiliary program is run which creates other files which in turn must be read back, in a second run of the LaTeX engine. So the run of an auxiliary program is always enclosed between two runs of a LaTeX engine.

Typically, the LaTeX run needs a LaTeX package associated with the auxiliary tool which performs reading and writing. An exception is `bibtex` and friends for which LaTeX engines support communication out of the box. An example with more complicated communication is `makeglossaries` with associated package `makeglossaries` which writes lines into the AUX file and which typically writes the main glossary into a GLO file. The tool `makeglossaries` which is invoked without ending, reads the AUX file, determines which other files to read, typically the GLO file also and writes the result into the GLS file. This is read back by the package `makeglossaries` in the next run of the LaTeX engine.

5.6.2 When running an auxiliary program

After the first run of the LaTeX engine, one must decide which auxiliary programs to run. For each auxiliary program, there is a specific file it reads or at least specific entries in a general file, typically the AUX file. If this file or these entries exist, the auxiliary program must be run and after the LaTeX engine must be rerun to read in the data created by the auxiliary program. As is discussed for each auxiliary program separately in Section 5.6.3, this file or these entries may change after each run of the LaTeX engine and as a result, the auxiliary program must be rerun as well. So, LaTeX engine and auxiliary program maybe must be run alternately.

Instead of checking whether the relevant data really changes, only the number of relevant lines and a hash is taken into account. This bears a minimal risk of not rerunning the auxiliary program although needed. Note that also package `rerunfilecheck` is based on hashes and bears the same risk.

It is an interesting detail, that deciding whether an auxiliary program must be run at all, i.e. for the first time, is just based on the existence of a specific file or of a specific line in a file, not comprising all pieces of information read by the auxiliary program. Nevertheless, if it is decided that the auxiliary program must be run, it is clear that the LaTeX engine must be run after also and so the information may

change. So one must be prepared for a rerun check. For this, all the information in the file(s) relevant for the auxiliary program must be hashed.

From the second run of the LaTeX engine on, only those auxiliary programs must be checked for rerun condition, for which a hash is present.

After these quite abstract considerations, let us apply these to the concrete auxiliary programs supported.

5.6.3 Why `rerunfilecheck` is not used for auxiliary programs

As described in Section 2.1, package `rerunfilecheck` is used to check whether the LaTeX engine must be rerun, and its authors also intended it to check for need of rerun of auxiliary programs. While this works satisfactory for a single index, it fails for multiple indices. Likewise, support for glossaries is buggy and works only in case of a single glossary, which in addition must be the main glossary. In contrast, the package `glossaries` supports multiple glossaries, with and without main glossary and even allows user-defined glossaries. It is awkward to implement rerun check for all this functionality with `rerunfilecheck`.

It may be surprising, that there are situations where even bibliography processors need to be rerun, among these backlinks, and citations in headlines and glossaries. Package `rerunfilecheck` does not take this into account. Accordingly, even `pythontex` may need a rerun, e.g. if code is executed in headlines or in captions of floating objects, because this may insert additional invocations and may change invocation order which may lead to different results.

While many auxiliary programs depend only on a subset of entries in their source file, `rerunfilecheck` can take files into account only as a whole. As a consequence, even if no rerun is required because the relevant entries did not change, `rerunfilecheck` could trigger useless rerun, because irrelevant entries in the relevant file changed.

Tanking all these aspects into account, we decided to provide an internal algorithm for rerun check of auxiliary programs, which is based on the ideas of `rerunfilecheck` but avoiding all its shortcomings.

Note also, that besides whether to rerun an auxiliary program, there is also the question in which case to run it at all, i.e. for a first time. Since package `rerunfilecheck` interprets a newly occurring file as a changed file, this case is addressed implicitly.

Unfortunately, not all packages associated with auxiliary tools give a hint if the auxiliary program must be run.

As described in Section 5.1, running a LaTeX engine 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, we call an auxiliary program. Similarly, the presence of code to be interpreted may be detected which is also written in a separate file and an external program, `pythontex` must be run to run the code in sequence and in many cases to determine the result of invocation.

In the next step, the LaTeX processor must read in the results of the auxiliary programs again to write bibliography, indices and glossaries and to insert the results of code invocations. Also, except the code invocations, all other pieces of information typically go into the table of contents. If code is invoked in a headline or in a caption, the result of the code invocation goes into the TOC and in the list of captions, e.g. the list of figures LOF also. So in any case, after an auxiliary program the LaTeX processor must be rerun.

Obviously, the run of a LaTeX processor may change page numbers and thus invalidate the index or the glossary. So the auxiliary program to create the index or the glossary must be rerun if the LaTeX processor changes the input file for the auxiliary program creating index or glossary and after that, the LaTeX processor must be run again.

What is less obvious is, that bibliographies may be invalidated also, e.g. because of a backlink or because a bibliographic reference occurs in a glossary. Even code may be invalidated by a run of the LaTeX processor if some code occurs in a floating object, e.g. in the caption or in a glossary. So code invocations may change order and also there may be additional code occurring not before later runs of the LaTeX processor. So also in this case, the according auxiliary program, `pythontex` must be rerun after the run of the LaTeX processor.

Summarizing, a run of the LaTeX processor may trigger invocation of each auxiliary program. This must be done if the according raw file changes. Note that various auxiliary programs share the AUX file to get information. So only the aspects relevant for the specific auxiliary program shall be taken into account. What makes things a bit more complicated is, that including TEX files yields included AUX files which must be taken into account also.

To implement rerun check completely reliable, huge parts of text files, a lot of information must be stored. Thus, we go a way like package `rerunfilecheck`, detecting only the change of number of relevant lines and the according hash. In extremely rare cases, this software may fail to rerun a program although needed, because number of relevant lines or its hash don't change although contents change.

Note that we only use the concept of `rerunfilecheck` to detect running and rerunning auxiliary programs, but we do not use the package `rerunfilecheck` itself for this task. This is because supporting all relevant auxiliary programs and also included AUX files would require considerable extensions on `rerunfilecheck`

and would impact considerable dependencies. So, as described in Section 5.7, `rerunfilecheck` is used to control rerunning the LaTeX processor as far as auxiliary programs are not involved, whereas detecting auxiliary programs to be rerun is done internally while the algorithm is inspired by the package `rerunfilecheck`.

5.7 Rerunning the LaTeX processor

CAUTION rework needed

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

As indicated in the previous sections, `latex2pdf` must be rerun, if an auxiliary program like `bibtex`, `makeindex` or `makeglossaries` had been run.

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 auxiliary programs like `makeindex` may be required and these also require another run of `latex2pdf` also to get the page numbers right.

The package `rerunfilecheck` allows detecting file changes via a hash almost for sure, and writes an according message into the LOG file. This is offered for pure rerun control of `latex2pdf` based on TOC, LOL, LOF and LOT, but also on the OUT file written by package `hyperref`. Partially, it supports also the need to rerun auxiliary programs, but for sake of uniformity, we refrain from using this, and rely on in internal algorithm also based on hashes.

Only for rerunning `latex2pdf` alone, we rely on package `rerunfilecheck`. This software just reruns textttlatex2pdf if it detects the pattern of warning written by `rerunfilecheck` into the LOG file.

Note that there are several packages which require additional runs, such as the package `longtable`, 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 may require a change in this software.

5.8 Checking reproducibility

There are use cases, where it is extremely important that the according artifacts are really reproducible. One is when we have to deliver the sources and the receiver has to reconstruct the artifacts. Another obvious use case is integration test for this software by ensuring that each artifact created is equivalent with a confirmed version, although this software changed. Details are given in Section 10.

Currently, reproducibility checks are supported for PDF files only. The problem with PDF files is, that besides visible contents they contain also metadata (see [PDF08] or [ISO20], each Section 14.3), which depends on the run of the conversion. For example the timestamp and the timezone of conversion goes into and derived from these other values.

There are two strategies to deal with the problem:

- Make the build process reproducible. The advantage of this approach 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.

- Use diff tools implementing a weaker notion of equivalence, in a sense visibility equivalence of some degree. One approach is the script `vmdiff` described in Section 3.5.7 which combines visibility equivalence with equivalence of part of metadata.

Since the first one works very well, it is the one we describe here, but it is always possible to configure a diff tool with a weaker equivalence check.

The first question is, whether reproducibility is requested. It is, if there is according magic comment in the LaTeX main file requires this as described in Section 3.1.1. If there is no such magic comment is present, if the setting `chkDiff` specifies so. If in this section settings are given without explicit reference, they are described in Table 6.13 on page 150 in Section 6.13.

Since date and time both visible and in the metadata of a PDF document is given relative to a timezone, for reproducible builds compilers must run with a fixed timezone and, as reproducibility shall not break if changing a timezone or if the country running the build changes between daylight saving time and standard time, we chose a uniform timezone namely UTC.

If a LaTeX main file is already under reproducibility control, then there is an according original PDF file in `diffDirectory` or in a subfolder to be compared

with a newly created PDF file which occurs in a subfolder of the TEX source directory `texSrcDirectory` described in Table 6.1 on page 122. The PDF file for comparison has the same path relative to `diffDirectory` as the created PDF file relative to `texSrcDirectory`.

First `pdfMetainfoCommand` is used to extract metadata `CreationTime` from the original PDF file. This comprises time and timezone which is UTC.

The compilation to create the new PDF file is run in an environment with that timezone and with that creation time. In addition, there is an environment variable forcing that the timestamp does not only affect metadata but also visual data of the PDF file to be created, as e.g. typically the date at the front page. Note that if the PDF file is created from TEX files via DVI/XDV files, both engines need the appropriate environment.

After creating the new PDF file with this environment, coincidence with the original PDF file is checked using the tool given by setting `diffPdfCommand` described in Table 6.13. If the actual artifact does not coincide with predefined one according to the chosen diff tool, a build exception is thrown as specified in Table 7.7.

If a LaTeX main file is not already under reproducibility control, then no original PDF file exists. In this case, the environment for compilation only ensures the timezone UTC. Then the created PDF file is copies at proper place into `diffDirectory` – that's all for setting a document under reproducibility control.

Finally, if a LaTeX main f8ile file is under reproducibility control but is to be changed in a way that also the according PDF file is affected, then before compilation just the original PDF file is deleted, and the workflow is as setting under reproducibility control.

Reproducibility is affected or even supported by various injections as defined in Section 3.5. First, the generic header described in Section 3.5.2 affects metadata, above all because it loads the package `hyperref`. Part of this metadata is overwritten by another header described in Section 3.5.4, to improve security and privacy, but enough metadata remains to keep up reproducibility. Reproducibility is guaranteed with the full set of metadata or with somehow reduced metadata. The only piece of information needed for reproducibility is `CreationDate` and this is preserved by the headers. Removing this also has severe consequences so that we can assume it is preserved. On the other hand, removing metadata may stabilize reproducibility as this is true for the banner which identifies the latex compiler and its version and consequently breaks reproducibility in any version change. Details to reproducibility with a focus on metadata are given in [Rei23b], Section 4.

Obviously, reproducibility checks cause work when putting a document under check, i.e. in the end phase of document development as defined in Section 3.6 or if the source document changes, i.e. if document development is entered again, or if

the output PDF changes unintended normally, although the sources did not change in an obvious way, which triggers again document development searching the cause of the change in the sources.

This LaTeX builder is not the tool for document development. Instead, Section 3.6.2 suggests to use `latexmk` for, and describes how `latexmk` is integrated in this LaTeX builder: This builder writes a config file `.latexmkrc` reflecting the settings of this software, at least to some extent. The config file `.latexmkrc` is again written as an injection and is described in Section 3.5.1. It supports reproducibility checks even reading magic comments, checking existence of original PDF file and reading its timestamp if the PDF file is present. Creation of the new PDF file takes timestamp and timezone into account.

Two further injections may be helpful in the context of reproducibility checks, both described in Section 3.5.6: `ntlatex` to create a PDF file and `vmdiff` realizing a weaker variant of diffing tool as described above: It checks for visual equality and equality of metadata.

For updating metadata only, we suggest the following technique: Keep the original PDF file in `diffDirectory` and check with `vmdiff` that visually, the PDf file remains the same and that the correct metadata is updated. Of course, a new timestamp is wanted. So in a second step, the original PDF file is deleted, compilation is repeated, e.g. by `ntlatex` and copied into `diffDirectory`.

There are rare occasions where the timestamp shall be set explicitly. This is not possible directly as it is read off from the original PDF file. We suggest to use `exiftool` to modify the `CreationDate` of the original PDF file in `diffDirectory` before compilation. This is done by something like

```
exiftool -PDF:CreateDate=2020-01-01T00:01:02Z xxx.pdf
```

Here, the option `PDF:CreateDate` is in fact the name of the tag to be written. Note that the timezone must be UTC represented by the `Z` signifying zero time offset compared to UTC. The attentive reader may wonder why the option is `PDF:CreateDate` instead of `CreationDate`. One may check with `pdfinfo`, that really `CreationDate` is modified. Note that `exiftool` writes the original PDF file into `xxx.pdf_original`

Two important details are not so obvious:

- Not only the given metadata is changed but also all metadata depending on it, in this case the trailer ID. This is to keep the PDF file consistent.
- The metadata is not really overwritten, but it is hidden by new metadata. In fact, `exiftool` uses incremental update specified for the PDF format, adding a layer describing the modification. All modifications done can also be undone by

```
exiftool -PDF-update:All= xxx.pdf
```

unless the PDF file has been linearized. LaTeX to PDF compilers always create linearized PDF files and never update incrementally.

To know that changing metadata is done by incremental update is important, insofar as a PDF file with modified timestamp and timezone differs from a PDF file compiled directly with the given timestamp and timezone; it is shorter. So, updating the timestamp of the PDF file in `diffDirectory` does not yield a PDF file which is reproduced. Compilation leads to another PDF file and only the updated timestamp is reproduced. This compiled PDF file is reproduced, so copying it the into `diffDirectory` solves the problem: Next compilation yields a PDF file with the correct timestamp and timezone, and it coincides with the PDF file in `diffDirectory`.

When subjecting a document under reproduction control with a predefined timestamp, then initially there is no original PDF file. One could place any PDF file in `diffDirectory`, overwrite the timestamp and timezone by `exiftool`. Is content is immaterial.

5.9 Alternative build process with `latexmk`

This section is on running the build process of LaTeX main files with `latexmk` or equivalent. Currently, that way only PDF files can be created. Although the functionality is readily explained, the intention is not so obvious: In Section 3.6.2 describes the role of `latexmk` as a build tool in the course of document development, whereas this LaTeX builder is for final, quality checked build. So the two tools seem to be complementary. Section 3.5.1 describes that this LaTeX builder can write its own configuration as a config file `.latexmkrc` for `latexmk` so that builds with `latexmk` are in line with final builds by this LaTeX builder itself internally.

So running `latexmk` from within this LaTeX builder seems superfluous at first sight. A closer look onto `.latexmkrc` unveils that this is just a Perl script which is very flexible realizing new or special functionality, whereas this LaTeX builder is tied to a quite rigid configuration in the pom. So, for example if for building a document tools are needed which are not supported by this LaTeX builder, their invocation can be implemented directly in `.latexmkrc`. Since this LaTeX builder writes a single `.latexmkrc` in the root directory `texSrcDirectory`, which must be made available in each subfolder by adding a link, the config `.latexmkrc` by this LaTeX builder may be replaced by a hand-crafted config file for each folder separately.

Another advantage being able to run `latexmk` from within this builder: It is conceivable, that the artifacts created in the course of document development using

latexmk cannot be reproduced by this builder. Most likely because .latexmkrc does not reimplement the internal functionality properly. Invoking latexmk in a final build reduces this risk to a minimum.

Further motivations for integrating latexmk in this builder, in particular for individual files: there are cases where the build process of latexmk works, but not the internal build process of this builder. Integrating latexmk offers the strengths of latexmk. Note that there are also cases where the built-in build process of this builder is mightier than that of latexmk. Another reason for integrating latexmk here, is the use case of source distribution: The document(s) may be passed to someone as the source, not as a target, like PDF. It is not clear that the "customer" uses this latex builder, but maybe (s)he uses latexmk. In this case it makes sense to check, whether the document can be built with latexmk alone.

Having explained this, the question arises why this LaTeX builder does not in general rely on latexmk and invokes LaTeX engines and other converters directly. One reason is that LaTeX builder does not only invoke converters, it also checks return values and, depending on the converter, log files emitting errors and warnings if appropriate. So, delegating to latexmk the user can no longer check that the build process passed without warning or error. A second aspect is, that the build algorithms differ: latexmk runs the LaTeX main file then detecting which files are missing and then tries to build these based on rules. The basic idea behind is "backward discovery" of dependencies, whereas this LaTeX builder first builds the graphic files globally (latexmk detects last) before for each LaTeX main file is compiled. So this LaTeX builder combines "forward discovery" and backwards discovery. Pure backward discovery is more elegant but as the LaTeX compiler stops at each graphic file not present before creating it and rerunning compilation of the LaTeX main file, it may result in excessive reruns of the LaTeX engine if there are many created graphics in the document.

So there are strong reasons to avoid latexmk, but there are also reasons to allow in special cases. The parameter $latexmkUsage described in Table 6.1 on page 122 allows gradually use of latexmk, not at all, fully or as backend where latexmk is invoked after graphic files have been created with an internal process. As a rule, latexmk shall be used as much as required and as little as possible.

This shows also, that it is a good thing to be able to activate latexmk in individual LaTeX main files which is realized with the magic comment latexmk. It can take the form latexmk=false, latexmk=true or just latexmk which is the short form of the latter. Magic comments are described in Section 3.1.1. In general, they overwrite settings. Here, the situation is a bit more complicated. Whereas $latexmkUsage allows three levels of usage, the magic comment can choose to use latexmk or not. If latexmk shall be used due to the magic comment, then it is used to compile the TEX file in any case, but it compiles graphic files only, if

`$latexmkUsage` takes the value `NotAtAll`. If `latexmk` shall not be used due to the magic comment, then it will never compile the TEX file itself, and if `$latexmkUsage` takes the value `Fully`, all required graphic files must be compiled for some reason, e.g. there is none to be compiled.

By the way, invoking `latexmk` from within this software is the same as invoking manually. Both are based on `.latexmkrc`. The features supported are described in Section!3.5.1. Among those are the supported targets, reading magic comments independently from internal implementations and support for reproducibility checks.

5.10 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!6.10. 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.

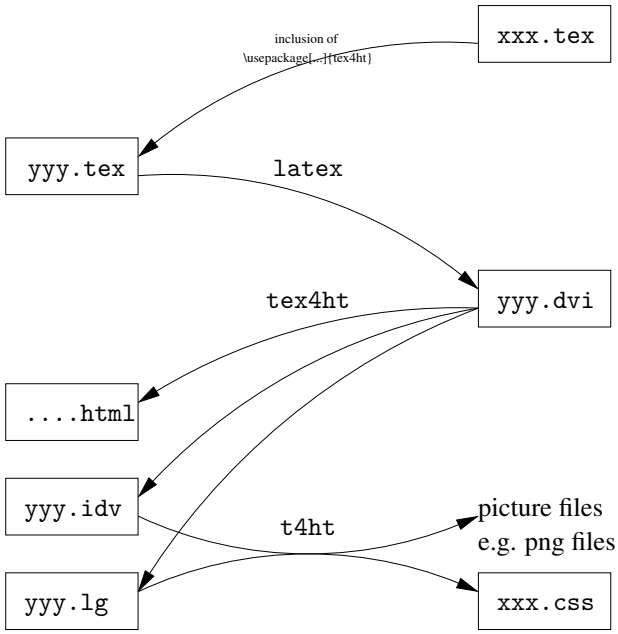

Figure 5.9: Conversion of a TEX file into an xml file

```
(/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^'
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
----------------------------------------------------------------------
```

5.11 Creating odt files

5.12 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 5.11. 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.

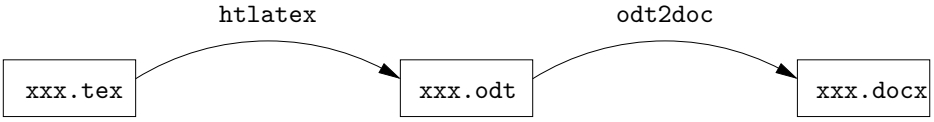

Figure 5.10: Conversion of a TEX file into a docx file

5.13 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.

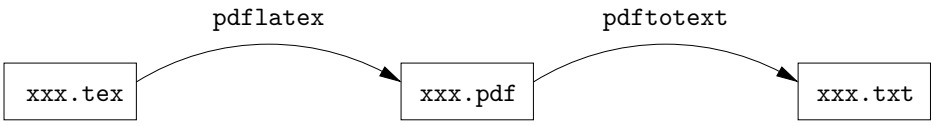

Figure 5.11: Conversion of a TEX file into a txt file

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 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. There are also general aspects, treated in Section 6.1. As this software mainly acts by invoking other converter and checker, most parameters refer to commands and options for invocations, but there are also parameters which cannot be associated to an individual invocation. Parameters referring to the conversion process or to checking as a whole are collected in Section 6.2. A special case is Section 6.3 which collects the parameters for goals `vrs` and `inj`. All the other sections refer to one or more converters.

The parameters are listed in Tables 6.1 through 6.13 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, only the tables 6.8 and 6.9 for `pythontex` and for `depythontex`, respectively, are collected in the single Section 5.5.

Table 6.1 shows parameters 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.

diffPdf A diff tool comparing the PDF file created with an expected PDF file. This is relevant only if a PDF file has been created and if the comparison is activated, which is not true by default.

It is a little more complicated with the parameters in Section 6.10.

The command name and the list of options describes the invocation of the command. This LaTeX builder supervises also the return value frequently and the log file is supervised.

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.

Besides the abovementioned patterns, describing events in log files, there are further patterns. The maybe the most prominent one, `patternLatexMainFile`, is devoted a separate section, Section 6.2.1. All patterns, i.e. all parameters of the form `patternZZZ` are interpreted as regular expressions in a variant slightly generalizing the default implementation for java. We owe description and implementation to Florian Ingerl.

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.

[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

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

This section describes the general parameters given in Table 6.1.

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 without blanks 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.
CAUTION: These targets are the default targets for any LaTeX main file, but depending on the document class, there may be further restrictions given by setting '`$docClassesToTargets`'. Currently, only the class `beamer` used for presentations has restrictions. Moreover, these targets may be overwritten for individual LaTeX main files using magic comments as described in Section 3.1.1.

`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 beginning of the contents of TEX-files which identifies a LaTeX main file, i.e. a file to be compiled. If the file is really a LaTeX main file, the pattern contributes to finding the targets for compilation. This may be done either directly via a magic comment or via the document class.
The default value for the pattern is chosen to match quite exactly the start of the LaTeX main files. Here we assume that the LaTeX main file should contain the declaration "`\documentclass`" or the old-fashioned "`\documentstyle`" preceded by a few constructs and followed by the document class. Among the few constructs are also comments and in particular magic comments.
Strictly speaking, a tight match is not necessary, only separation of LaTeX main files from other files is and so is extraction of the document class. For a more thorough discussion, and for an alternative approach, consult the manual.
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. In any case, matching of the group named `class` must be retained so that the document class is matched.

`docClassesToTargets` see description below

Assigns to document classes their allowed '`targets`'. The map expression is a list of chunks separated by a single blank. Each chunk is divided by a single colon in a comma separated list of document classes, and a comma separated list of targets.
A chunk means that all given document classes are compiled for the given targets. Thus, the set of document classes may not be empty, i.e. the colon may not be at the first place of its chunk. In contrast, a colon at the last place of a chunk indicates an empty target set, meaning that documents of the given class are not processed at all.
The document classes of the chunks may not overlap. A document of a class is compiled for a target if this is specified so by a chunk.
As a side effect, compilation of document classes cause warnings if not registered here. The default value consists of two chunks:

- '`article,report,book,minimal:chk,dvi,pdf,html,odt,docx,rtf,txt`' ensures that article and book and others allow all targets.
- '`beamer,leaflet,scrlttr2:chk,pdf,txt`' beamer allows mainly `pdf` and derived from that `txt`. Checking with `chk` does not depend on the document class. Note that maybe leaflets or letters may work for DVI or XDV also, even for word formats and related, we restrict ourselves to the given output for simplification.

CAUTION: Due to a bug in maven, setting this to the empty string is ignored.
CAUTION: This setting is ignored, if targets are specified for individual LaTeX main files using magic comments as described in Section 3.1.1.

`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`.

`latexmkUsage` `NotAtAll`

The extent to which `latexmk` or to be more precise, the command given by `latexmkCommand` is used to build. The following values for build strategy are allowed:

NotAtAll `latexmk` is not used at all.

AsBackend `latexmk` is used as backend, i.e. graphic files are created as for goal `grp` as in strategy `NotAtAll` before `latexmk` is invoked on the individual LaTeX main files.

Fully build is by applying `latexmk` on the individual LaTeX main files without any prior actions.

This setting can be overwritten for individual LaTeX main files by the magic comment `latexmkMagic` described in Section 6.2.1.
For a more detailed description of usage of `latexmk` see Section 5.9.

`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 LaTeX 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

6.2.1 The parameter `patternLatexMainFile`

Before reading the details given in this section, the user is advised to at least skim through Section 3.1.1 and 3.2 for intuitive understanding.

The regular expression pattern `patternLatexMainFile` matches exactly the files to be compiled, the so called LaTeX main files, and for LaTeX main files it extracts the following pieces of information, all by named capturing groups which have the form `(!<name>pattern)` but in the pom `<` and `>` must be escaped and so capturing groups take the form `(!&lt;name&gt;pattern)`. The capturing group named `docClass` extracts the document class from the command `\documentclass`. If `patternLatexMainFile` matches, also the capturing group `docClass` matches, so that for LaTeX main files the document class is always known. All other capturing groups are defined though magic comments. As the document class, they are directives specific to the given file on how to compile it. They override the general settings given in the pom. A magic comment may not match, which means that there is no according specific directive and so the general setting holds.

Whether a capturing group must match or not, the regular expression pattern `patternLatexMainFile` must contain each of the following named capturing groups, because the software asks for it. Distinguish between the pattern and the matching strings: Whereas in the pattern all groups must be mentioned, a string may match without matching the group. Whereas `docClass` is mandatory, i.e. the according group matches, the magic comments are all optional, i.e. they need not match any part of the string.

docClass the document class given by the command `\documentclass`.

programMagic the LaTeX engine to be used specifically for the according document. This is intended to be specified only if the required engine for the given document deviates from what is specified globally as setting `latex2pdfCommand` described in Table 6.4 on page 134.

targetsMagic the targets to be built. This is intended to be specified only if the targets for the given document deviate from what is specified globally as setting `targets` and `docClassesToTargets`, both given in Table 6.1 on page 122.

latexmkMagic, latexmkMagicVal whether *for creating PDF files* `latexmk` shall be used. This is intended to run the build process with `latexmk` although the global setting `latexmkUsage` given in Table 6.1 on page 122 may specify direct compilation without `latexmk`.

The magic comment can take the form `% !LMP latexmkMagic=<bool>` or `% !LMP latexmkMagic` which is just short for `% !LMP latexmkMagic=true`.

chkDiffMagic, chkDiffMagicVal whether the *created PDF file* shall be checked against an original ensuring that it is correctly reproduced. This is intended to control a check specific for this file overwriting the general setting `chkDiff` given in Table 6.13 on page 150.

The magic comment can take the form `% !LMP chkDiff=<bool>` or `% !LMP chkDiff` which is just short for `% !LMP chkDiff=true`.

The default pattern for identifying LaTeX main files and to extract the above pieces of information is given by Listing 6.1.

```
1 \A
2 (%\s*!\s*T[eE]X (TXS|spellcheck|encoding|root).*\R)*
3 (%\s*!\s*T[eE]X program\s*=\s*(?&lt;programMagic&gt;[^} ]+)\R)?
4 (%\s*!\s*T[eE]X .*\R)*
5 (%\s*!\s*LMP (?&lt;chkDiffMagic&gt;chkDiff)(=(?&lt;chkDiffMagicVal&gt;true|false))?\R)?
6 (%\s*!\s*LMP (?&lt;latexmkMagic&gt;latexmk)(=(?&lt;latexmkMagicVal&gt;true|false))?\R)?
7 (%\s*!\s*LMP targets=(?&lt;targetsMagic&gt;(\p{Lower}|,)+)\R)?
8 (\s*(
```

```
9  \\RequirePackage\s*(\[(\s|\w|[,=])*\])?\s*\{(\w|-)+\}\s*(\[(\d|[-./])+\])?|
10 \\PassOptionsToPackage\s*\{(\s|\w|[,=])*\}\s*\{(\w|-)+\}|
11 \\newbool\s*\{\w+\}|
12 \\setbool\s*\{\w+\}\{(true|false)\}|
13 \\DocumentMetadata(?&lt;brace&gt;\{(?:[^{}]|(?'brace'))*\})|
14 \\input\s*\{[^{}]*\}
15 )?\s*(%.*)?\R)*
16 \\(documentstyle|documentclass)\s*(\[[^]]*\])?\s*\{(?&lt;docClass&gt;[^} ]+)\}
```

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

Let us trace through line by line:

1 The `\A` indicates the start of the file.

2–4 These lines match magic comments `% !TEX`, which are used by other build tools also. Line 3 extracts `programMagic` from the first magic comment of the form `% !TEX program=…`. This is the behavior of the other tools also. The other lines are to skip information from magic comments `% !TEX` which are not needed.

5–7 These lines match magic comments of the form `% !LMP…` which are specific for this software. Like the above magic comments they are all optional, but their ordering is fixed:

`chkDiffMagic` to activate diffing to check reproduction, or in conjunction with `chkDiffMagicVal` to switch reproduction check.

`latexmkMagic` to delegate build to `latexmk`.

`targetsMagic` allows to specify a list of targets. This is the sole of these magic comments not only applying to creating PDF files.

8–11 This defines material which may precede the command `\documentclass`, except for magic comments and is the only one without magic comments. Besides lines with specific commands, it matches empty lines and comment lines. Also, a line may start with whitespace and may contain a comand and end in a comment. The commands specified there may occur in arbitrary multiplicity and order. This section is likely to be modified by the user.

11 Matches the command `\documentclass` and extracts `docClass`.

Between magic comments and `\documentclass` or `\documentstyle` only the following material is allowed:

- the command `\RequirePackage` specifying packages to be loaded before `\documentclass`, in contrast to `\usepackage` which is used after,

- the command `\PassOptionsToPackage` allowing to pass one or more options to a package, although including with `\usepackage` is without options,
- `\newbool` and `\setbool` to define and set a boolean value defining variants (preceeded by `\RequirePackage{etoolbox}`),
- the command `\DocumentMetadata` allowing arbitrarily nested braces,
- the command `\input`, and
- whitespace, empty lines, comment lines even magic comments, although for this tool they are ignored.

This may be too restrictive and here is the point, where the use has freedom to change the pattern. On the other hand, `\input` offers a quick workaround to add material if a user is not familiar with regular expressions.

In the long run it must be thought of weakening the pattern: It is not necessary, that exactly the correct files are parsed, because incorrect files are detected by the LaTeX engine anyway. Instead, among the correct files the LaTeX main files shall be detected.

As a workaround for very special LaTeX main files, it is a good idea to let it indicate in a magic comment. Then the pattern as a whole must match, even not matching a `\documentclass`. From the point of view of this software, it makes sense to specify the document type in the magic comment then. Thinking one step further, also specifying the target or the LaTeX engine in a magic comment indicates already a LaTeX main file. Whereas the target set makes the document class superfluous, this is not the case for the magic comment specifying the LaTeX engine.

The LaTeX extension LaTeX workshop for VS Code offers two similar alternatives to identify LaTeX main files: Occurrence of `\documentclass` without checking preceding material and absence of first line `% !TEX root=` declaring a TEX file as depending on a LaTeX main file which must be given explicitly. The first alternative risks that a TEX file is recognized as main file, just because it deals with document classes, whereas the second alternative is inconvenient and does not work if a file has two potential LaTeX main files as is suggested for `beamer` presentations in Section 3.1 and realized in [Rei23a]. Although presence of `% !TEX root=` indicates that the according file is no LaTeX main file, this software ignores this magic comment.

Emacs with package AUCTeX, uses an alternative to the current technique to determine the LaTeX main files: 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:
```

Unlike the document class to be extracted from `\documentclass` and unlike other magic comments to be taken into account, those of AUCTEX are at the end of the file.

Although the author considers this approach charming, this software ignores AUCTEX-style magic comments, since otherwise the whole file is to be parsed. Sticking to regular expressions, the parsing engine must then keep the whole file in memory. All this would push down performance.

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 goals vrs and inj

This section describes the parameters for the goals vrs and inj given in Table 6.2. As illustrated in Listing 2.4 of the part of the pom referring to this plugin, in general parameters are configured in a settings element contained in a general configuration element. In contrast to this, the parameters for the goals vrs and inj are given in configurations within executions specific for these goals.

Parameter Explanation	Default
`versionsWarnOnly`	`false`
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.	
`injections`	`latexmkrc,chktexrc`
Indicates the files injected by the goal `inj`. This is a comma separated list of `injections` without blanks. For further description see Section 3.5.	

Table 6.2: Parameters for goals `vrs` and `inj`

6.4 Parameters for graphical preprocessing

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

TODO: do this.

Parameter Explanation	Default
`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 `$gnuplotCommand`'s output terminal "cairolatex", used for mixed latex/pdf-creation. Note that the option '`pdf|eps`' of the terminal '`cairolatex`' is not available, because it is set internally.	
`metapostCommand`	`mpost`
The command for conversion of gnuplot-files into metapost's postscript.	
`metapostOptions`	see Section 6.4.1
The options for the command `$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 6.4.2.	
`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

6.4.1 The parameter `metapostOptions`

The options of the (sole standalone) metapost compiler are given in the metapost manual [Hob24], 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 allow dependencies tracking.

-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 [Hob24], 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.

6.4.2 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.

6.5 Parameters for the LaTeX-to-pdf Conversion

This section describes the parameters of the LaTeX engine 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, i.e. the LaTeX engine. Further formats are DVI and XDV and also other formats based on these. Expected values are `lualatex`, `xelatex` and `pdflatex`. CAUTION: This setting may be overwritten for individual LaTeX main files using magic comments as described in Section 3.1.1.
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 6.5.1
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 6.5.2
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 6.5.3 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 conversion

6.5.1 The parameter `latex2pdfOptions`

An overview over the options supported by the usual latex engines in distribution TeX Live is given in [Rei23b], Section 2. In particular, there is a table with the options occurring in any LaTeX engine and columns indicating for each option for which engines 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 engines. 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 engines, 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 engines are double dash. It must be clarified, whether they understand single dash. If not one has to clarify whether in TEX Live all engines understand double dash. If so all must be changed into double dash.

6.5.2 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 starting with `pdfTeX warning` and being specific for the compiler `pdflatex`,
- Warnings on inclusion of a PDF file, e.g. inclusion of PDF files with incompatible version, starting with `warning (file <filename>) (pdf inclusion)`,
- Font specification warnings starting with `* fontspec warning`[2],
- Further warnings not identifying themselves as warnings as the word "warning" does not occur. Still they are treated as warning because they all indicate some imperfection in the output.

The resulting default pattern is

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

6.5.3 The parameter `patternReRunLatex`

TODO: rework based on comments in class Settings.

For the package `rerunfilecheck` an analysis of the code is possible, and the warnings emitted by this package indicating the need for rerun are taken into account for the pattern.

Besides this package, also other packages may require rerun, but these are not analyzes systematically. A first step would be to analyze those given in `header.tex` created by injection.

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.

The resulting default pattern is

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

[2]Please note the leading character "*".

```
LaTeX Warning: Etaremune labels have changed\.$|
\(rerunfilecheck\)               Rerun to get outlines right$|
\(rerunfilecheck\)               Rerun LaTeX)
```

There is one `Info` message in there, also indicating the need for rerun. This is inserted because another rerun warning may fail to apply because it contains the file name and if this is too long, then the required sequence "`Rerun.`" is cut off and is not on the current line.

Still what is good, if such a warning is not recognized as a pattern indicating the need for rerun, it occurs in the final LOG file and is recognized as a warning. So it is merely impossible to get a result with not enough reruns and without warning.

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

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

Table 6.5: The BibTeX-utility

6.7 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` The MakeIndex command to create an ind-file from an idx-file logging on an ilg-file.	`makeindex`
`makeIndexOptions` The options for the MakeIndex command.	the empty string
`patternErrMakeIndex` The pattern is applied line-wise to the ilg-file and matching indicates that `$makeIndexCommand` failed. The default value is chosen according to the '`makeindex`' documentation.	`(!! Input index error )`
`patternWarnMakeIndex` The pattern is applied line-wise to the ilg-file and matching indicates a warning `$makeIndexCommand` emitted. The default value is chosen according to the '`makeindex`' documentation.	`(## Warning )`
`patternReRunMakeIndex` **This parameter is deprecated since version 2.1.** Rerun check of auxiliary programs do not read the LOG file. Details of the present algorithm are described in Section 5.6. 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.	
`splitIndexCommand` The SplitIndex command to create ind-files from an idx-file logging on ilg-files. This command invokes `$makeIndexCommand`.	`splitindex`
`splitIndexOptions` The options for `$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.	`-V`

Table 6.6: The utilities MakeIndex and SplitIndex

6.8 Parameters for creation of the Glossary

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 `$makeGlossariesCommand` failed. The default value '`( unable to execute: )`' is chosen according to the `makeindex` documentation. If the default value is not appropriate, please modify and notify the developer of this plugin.	`(^\*\*\* unable to execute: )`
`patternErrXindy` The pattern in the GLG (`makeglossaries` log file)-file which indicates an error when running '`xindy`' via `$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`	

This parameter is deprecated since version 2.1. Rerun check of auxiliary programs do not read the LOG file. Details of the present algorithm are described in Section 5.6. 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.

Table 6.7: The MakeGlossaries-utility

6.9 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 Explanation	Default
`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 PLG (`pythontex` log file: home-brewed since the original application does not write log files)-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 `$pythontexCommand`. For the possibilities see the manual of the pythontex package or the help dialog of `pythontex`. CAUTION: `--rerun` and `--runall` cannot be specified both in one invocation. In the context of this software, the option `--interactive` is not appropriate. CAUTION: For many options of the command line tool, there is an according package option and the latter overrides the former. CAUTION: This software overwrites settings `--rerun` and `--runall` anyway, and forces setting `--rerun=always`. The default value is `--rerun=always`.	
`patternErrPyTex`	see Section 6.9
The pattern in the PLG-file indicating that running `pythontex`, resp. `pythontexW` via `$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 6.9
The pattern in the PLG-file indicating a warning when running `pythontex`, resp. `pythontexW` via `$mpythontexCommand`. If this is not appropriate, please modify and notify the developer of this plugin.	
`prefixPytexOutFolder`	`pythontex-files-`
The prefix of the name of the folder written by `$pythontexCommand`. The full name of that folder is this prefix followed by the jobname of the LaTeX main file, i.e. the filename without ending. CAUTION: This is readonly, because in both, the pythontex tool and the according LaTeX package this prefix is hardcoded at time of this writing.	

Table 6.8: Injecting output of code via `pythontex`

Parameter	Default
Explanation	
`depythontexCommand`	`depythontex`
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 DPLG (`depythontex` log file: home-brewed since the original application does not write log files)-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`.	
`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\(s\)
```

6.10 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 6.10.1

Table 6.10: The LaTeX-to-html-converter

6.10.1 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.

6.11 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	Default
Explanation	
`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`.

6.12 Parameters for 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, both given in Table 6.12.

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.

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

6.13 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 7.7 is thrown.

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

Parameter Explanation	Default
`diffDirectory`	`src/main/resources/docsCmp`
Diff directory relative to `$baseDirectory` used for diffing actually created artifact against prescribed one inthis directory. This is relevant only if `$chkDiff` is set.	
`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`. If this is set, the system time is set to 0 indicating 1970–01–01. Note that currently, only pdf files are checked. This setting can be overwritten for individual LaTeX main files by the magic comment `chkDiffMagic` described in Section 6.2.1. 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.
`pdfMetainfoCommand`	`pdfinfo` Command to retrieve metainfo from PDF files. Essentially, there are two possibilities, `exiftool` or `pdfinfo` but currently this software is restricted to the latter. At time of this writing, only creation time is considered. Note that meta info `CreationTime` is not identical with creation time in a file system.
`pdfMetainfoOptions`	`-isodates` The options for the command `$pdfMetainfoCommand` which is currently always `pdfinfo`. At time of this writing, only creation time is considered. This software has little flexibility in treating various time formats, so it must be decided. Format offered by `pdfinfo` most commonly known and easily converted to the required epoch time, is really according to ISO 8601. This motivates `-isodates` to be a mandatory option. Further options do not make sense, as currently only creation time is used. So `-isodates` is more than a mere default value.

Table 6.13: The parameters of the pdf differ

6.14 Parameters for `latexmk` and related

As described in Section 5.9, based on the parameter `$latexmkUsage` described in Table 6.1 on page 122, the build process can be delegated gradually to `latexmk` or an equivalent tool. Table 6.14 lists the parameters controlling invocation. Note that besides the options, which shall be used with care, also the config file `.latexmkrc` goes into. The details concerning the config file are described in [Col23], Section "CONFIGURATION/INITIALIZATION (RC) FILES". On the other hand, as indicated in [Col23], Section "DEALING WITH ERRORS, PROBLEMS, ETC", `latexmk` does not write its own log file and so there is no parameter in Table 6.14 for a pattern of warnings or errors.

Parameter	Default Explanation
`latexmkCommand`	`latexmk`

	The latexmk command to create a pdf-file from a latex file and other files. Instead of the default value `latexmk` a wrapper is conceivable, a reimplementation seems quite unlikely *smile*.
`latexmkOptions`	empty The options for the command `$latexmkCommand`. Since this command is controlled to a wide extend by the config file `.latexmkrc`, the options are of minor importance. On the other hand, there are options not allowed for this software because they change behavior in a way not taken into account. So add options with care. The allowed options and their defaults are given in [Col23], Section "LATEXMK OPTIONS AND ARGUMENTS ON COMMAND LINE".

Table 6.14: The parameters for `latexmk` and related

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 '$outputDirectory' should be a directory if it exists, but is not.` The output directory is given in the pom/build-file with default value `./target/site/..` The output directory is where the result of the goal/-task are copied to. If it does not yet exist, it is created but if it exists and is a regular file, it cannot be created anymore.
`TSS04`	`The target set '$targetsStr' contains the invalid target '$targetStr'.`

Indicates that a target `$targetStr` in a target set given in a context `$context` is unknown, e.g. because it is misspelled. The context is either the setting `$targets` or the target set in a chunk of setting `$docClassesToTargets` or in a magic comment specifying `$targets`.
For a description of the settings see Table 6.1 on page 122. See also the Exception `TSS11` in this table.
For each target, there is an according goal and so it 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 maven's 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.

```
TSS11      The target set '$targetsStr' in $context
           repeats target '$target'.
```

Indicates that a target `$targetStr` in a target set given in a context `$context` is repeated, despite sets contain elements only once. The context is either the setting `$targets` or the target set in a chunk of setting `$docClassesToTargets` or in a magic comment specifying `$targets`. For a description of the settings see Table 6.1 on page 122. See also the Exception `TSS04` in this table.
`TSS12` `Invalid mapping '$chunk' of document classes to targets.` Indicates that the chunk `$chunk` in parameter `docClassesToTargets` is syntactically not allowed. For a description of the syntax see Table 6.1 on page 122.
`TSS13` `For document class '$cls' target set is not unique.` Indicates that in parameter `docClassesToTargets` a class defines its targets more than once.

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 `$chkDiff` described in Section 6.13 is set. Then the diff command `$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 Add file '$pdfFileCmp' to compare with artifact '$pdfFileAct'!`
The PDF file `$pdfFileCmp` expected for comparison with the PDF file `$pdfFileAct` created from a LaTeX main file does not exist. It is expected only if a diff check is configured according to `$chkDiff` described in Section 6.13. This warning is normal if the document is added newly. Then just copy the created PDF file (maybe preserving modification time) after quality check. This warning is also normal if a document is actively modified. Then before building the file `$pdfFileCmp` shall be removed before compilation to force this software to assign a new timestamp, e.g. into first page and metadata. 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 '$logFileName'; may hide warnings/errors.`
FIXME
`WLP02 Cannot read log/aux file '$logFileName'; $kind may require rerun.`

FIXME

`WLP04` `Cannot read idx file '$idxFileName'; skip creation of index.`

FIXME

`WFU03` `Cannot close '$closeable'.`

FIXME

`WFU04` `Could not assign timestamp to target file $file.`

Currently NOT USED!

The former explanation was as follows If either the parameter '$chkDiff' described in Table 6.13 on page 150 is set or the magic comment `chkDiff` described in Section 3.1.1 occurs, then the modification time of target files must be set explicitly. In this situation, this warning occurs if setting the modification time could not be set.

`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.

```
WLP09 For file '$texFile' targets are neither specified by magic comment
      nor restricted by document class '$docClass'.
```

Indicates that the LaTeX main file `$texFile` has neither a magic comment specifying the targets nor for the document class parameter `docClassesToTargets` described in Table 6.1 specifies the allowed targets. Since no restriction on targets are known, `$texFile` is compiled for all targets given in `$targets` given also in Table 6.1. To avoid this warning, just add `$docClass` to `docClassesToTargets` or specify targets by magic comment.

```
WLP10 Degraded identifier for '$file';
      augmented risk not to rerun although necessary.
```

Indicates that an auxiliary file `$file` which is used to determine whether an auxiliary program shall be rerun could not be completely evaluated. An example of an auxiliary file is an IDX file. If it changes not only `makeindex` but also the LaTeX compiler need to be rerun.

A special kind of auxiliary files are AUX files. They may be used to create bibliographies or glossaries. They are special in that they may include other AUX files, namely those corresponding with included TEX files. In this case, `$file` is the top level AUX file.

Table 7.10: Miscellaneous errors and warnings

FIXME: to be added.

Chapter 8

Gaps

This chapter collects some gaps, but not all and sorts them into categories.

8.1 Gaps in graphics

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 GIF (Graphics Interchange Format, allows also animations)-format but maybe a converter to PNG is all needed.

8.2 Build mechanism

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 pv itself, once for ODT and once for DOCX.

8.3 Indices

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.

8.4 Glossaries

According to [Tal24b], Section 1.3, there are various options to create a glossary, whereas this software supports option two only described in Section 1.3.2, which uses `makeindex` for indexing. Currently, indexing with `xindy` is not supported. The last two options are available only with package `glossaries-extra` which this software will support in later versions.

By default, package `glossaries` creates a single "main glossary", which can be switched off specifying the option `nomain` described in Section 2.6. In this case at least, more specific glossary types must be specified. This can be done by options like `acronyms` described in Section 2.7 or the `symbols`, `numbers` or `index` options described in Section 2.9. As the `index` option collides with indexing as performed by this software, the option `index` is not allowed.

The package `glossaries` itself supports new glossary types via the command

```
\newglossary[log-ext]{name}{in-ext}{out-ext}{title}[counter]
```

described in [Tal24b], Section 9. In fact, the glossary types accessible via options and even the main glossary are defined internally that way.

Although the glossary algorithm of this software, in particular rerun management as described in Section 5.6 can create any kind of glossaries created with `\newglossary`, and it can also clean up files created in conjunction with glossaries as long as the file endings do not contain ".", defining new glossary types is not recommended because `latexmk` cannot mimic this with a fixed `.latexmkrc` file, neither in creation rules nor in patter for files for deletion and because collision, e.g. with indexing, cannot be excluded.

Reading [Tal24b], Section 13.1, the glossary option `index` seems to allow creating indices through the `glossaries` package making any index-package obsolete. This software does not support that technique offered by the package `glossaries`.

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 `vscode`, 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 creating 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 LaTeX engine) 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 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 files: Documentation: shall not include the environment `documentclass/documentstyle` in an input. Also check whether command `RequiresPackage` makes sense 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 6.5.3 needs more careful investigation. This is done to some extent in class `org.m2latex.core.Settings`.

Chapter 10

Preferred usage, Test Concepts and Tests

This software may be used in different environments, is highly configurable and also there is a huge amount of packages potentially in use.

In order not to get lost in extensive tests for covering all and everything, the author applies the notion of *preferred usage*. This is essentially the way the author uses this software. This is also what is tested extensively. Other ways of usage are supported insofar as reported bugs are fixed in general, but since explicit tests lack, the quality is lower for these cases.

The preferred usage is defined as follows:

- Linux, to be more precise, SuSE tumbleweed. The author used this software frequently and always with success on Windows also. As a shell use git shell.
- LaTeX Distribution TeX Live, to be more precise the SuSE specific variant. In the long run MiKTeX must be at least tried also. As MiKTeX is available for Linux also, test will be under Linux.
- VS Code with the extensions installed by `instVScode4tex.sh` which is described in Section 3.5.5, and the viewer `okular`. This is defined here, although not going into the artifacts.
- The VS Code extension `james-yu.latex-workshop`, which is installed by `instVScode4tex.sh` is used only with build recipe `latexmk (latexmkrc)`.
- The maven plugin, rather than the ant task.
- Configuration is the default setting of this plugin. In particular, the latex processor is `lualatex`.

- If `latexmk` is used, then with config file `.latexmkrc`, whereas `chktex` is definitely used, and it is configured with config file `.chktexrc`. Both config files are injected by `inj` described in Section 3.5.1.
- Neither `latexmk` nor `chktex` run from the command line is used with options. This is the default option list for `latexmk` but not for `chktex`.
- Compilation with `latexmk` but not by setting which is by default

  ```
  <latexmkUsage>NotAtAll</latexmkUsage>
  ```

  but activated by the magic comment

  ```
  % !LMP latexmk
  ```

- Reproducibility checks both by setting although is by default

  ```
  <chkDiff>false</chkDiff>
  ```

  and also activated by the magic comment

  ```
  % !LMP chkDiff
  ```

  Note that this is exceptional in that the preferred usage deviates from the defaults: Tests are based on reproducibility check setting `chkDiff` to true and there are many internal projects where reproducibility checks are selectively triggered by the magic comment.
- Document classes according to preferred usage which is by default are `book`, `article`, `beamer`, `leaflet`, `scrlttr2` and `minimal`. For beamer documents, preferred usage are both, the presentation and the handout as described in Section 3.1.1. Note that, unlike most other aspects of preferred usage, this is not tested through this manual but with the beamer presentation and handout given by [Rei23a]. Also, the test for `leaflet` class is tested by document [Rei24a] and the test for the letter class is tested in [Rei24b]. The `minimal` document checking the according document class has no reference.

  Caution: Currently, the classes `letter` and `report` are not preferred usage, but the latter is accepted with default configuration without warning. In the

long run, the manual shall be a report, whereas there shall be a user guide which shall be a book.

Among the document classes provided by KOMA script described in [Koh23], only `scrlttr2` is preferred usage. It replaces `letter`.

- LaTeX packages are those given by `header.tex` described in Section 3.5.2.
- Graphics are in the formats described and used in the manual.
- Using tools pythontex, makeindex, splitindex, makeglossaries belongs to the preferred usage. This does not mean that these tools must be used, but it means that the usage is restricted to these tools, not taking graphics into account.
- Tools must be used with accepted version, in the sense that `mvn latex:vrs` does not emit a warning.
- The output format is PDF.
- Security issues are addressed by including a header described in Section 3.5.4.

It is the "founded conviction" of the author, that in most of the use cases, restriction to the preferred usage is possible but when deviating, there is some increased risk that there is a bug in this LaTeX builder.

The set of documents coming with this software are compiled sticking to the preferred usage. Above all, this manual [Rei] not only describes all vital features, but also uses them, with one big exception: Its document class is `book`, and it cannot have other document classes of course. Most other documents are `article`s and [Rei23a] refers to a presentation with class `beamer` and to the according handout which is again an article. So the build process for these documents altogether cover the preferred usage to a wide extent. Thus, a bug in this LaTeX builder is likely to be reflected in a deficiency in the compiled version of one of these documents.

This shows that testing the compiled documents is a reasonable test strategy. It is not feasible to do this manually for whole documents, and it is also technically close to impossible to do it automatically. What can be checked automatically is coincidence with the last document.

So the strategy is to either change the (source of the) software or the *source of the* documents, but never both at once. If the software is changed, the created documents must persist. Changes in the manual are locally in the sources and result in local changes in the compiled document, because the software was not changed. Thus, the compiled document can be checked manually. Since the only output format being part of the preferred usage is PDF, only compilations into PDF must be taken into account.

Section 5.8 describes how this LaTeX builder can perform an equality check on PDF documents. There, both Section 6.13 on parameters for equality check is referenced and Table 7.7 comprising build failures if the documents do not coincide. We highlight the parameter `chkDiff` which determines whether the check is performed and build failures `TLP01` thrown if the documents differ. In tests, `chkDiff` is set `true`, the default is `false`.

Although this test concept seems appealing, it is not always easy to realize.

Before explaining the difficulties, let us differentiate between the two ways the pom of this plugin uses this plugin itself. The pom for performing tests is based on pom4pdf.xml, not on the actual `pom.xml` of the project a version of which is on github. For `pom4pdf.xml`, the current version is determined by filtering, which remains correct even during the release process. In addition, the project `pom.xml` used for development contains another, explicit dependency to `latex-maven-plugin`. This one is used for creating the documents for the site and also for developer tests. Thus, during development `pom.xml` is kept close to `pom4pdf.xml`, and it has a snapshot version `x.y.0-SNAPSHOT`.

Let us first consider the case of development in which the version of this plugin is a SNAPSHOT version. Then tests refer to the (snapshot) version under consideration. If a change is made to the documents and all tests pass, the changed documents are compiled with current software, and go into the next snapshot deployed. To be precise, the documents are compiled with deployed software, which is equivalent with the software compiled from the current sources.

If in contrast the software is changed, keeping the manual unchanged, then still compilation of the documents and also check is performed with the deployed version of this `latex-maven-plugin`. So, to decide whether the documents remain the same after the software change, a second build must be performed, because this compiles with the newly deployed snapshot of this software.

The situation is even more complicated if development is finished for the current version and a new release must be built. As is state of the art, for this task the `maven-release-plugin` is used. It requires for sake of reproducibility, that the pom of the project, not the pom has dependencies and plugins only in release versions, no snapshot versions.

As is state of the art, for release the `maven-release-plugin` is used. Whereas it has no explicit restrictions on the pom for tests `pom4pdf.xml`, it requires for sake of reproducibility, that the pom of the project `pom.xml` has dependencies and plugins only in release versions, no snapshot versions. This applies also to this plugin. For development, it has version `x.y.0-SNAPSHOT` to deploy `x.y.0`, and this is also the version one wants to create a site with, but this is the one to released at present. A possible fallback is always to deactivate the usage of this plugin. As a consequence, later a version `x.y.1` shall be released, which uses `x.y.0` for

site creation. Better is to use the last release version and to configure it so that the documents can be compiled with the old version. This may require a creative release planning, including features used to compile documentation and maybe a change in the parameters or some other change in the environment, which must be compensated in later releases also.

Let us give examples of creative realizations of the described test concept relying on thorough release planning. To release `2.0.0` starting with the prior version `1.8.0`, almost only injections are added. These can be done manually using `2.0.0-SNAPSHOT`. Then the injected files are checked in into version control and then the documentation can be compiled with old version `1.8.0` with the same result. In a release `2.0.1`, version `1.8.0` can be replaced by `2.0.0`.

For version `2.1.0` it is planned, that this plugin can use `latexmk`, and in the manual this is also described. In `2.1.0-SNAPSHOT` the manual may be compiled using `latexmk`, but nevertheless, in `2.1.0` the manual is still compiled without `latexmk`, using release `2.0.0` for creating the manual for the site. Only in `2.1.1`, also the manual is included in the site using `latexmk`.

For version `2.2.0` it is planned, to support `bib2gls` directly. Observe that `2.1.0` supports can treat `bib2gls` via `latexmk`, but without all the monitoring `2.2.0` offers with direct support. Of course, the manual describes direct support and some `2.2.0-SNAPSHOT` is able to compile the manual using `bib2gls` directly. Nevertheless, close to release for site creation, `2.1.0` is used again relying on `latexmk` to invoke `bib2gls`. In `2.2.1` then `2.2.0` can be used for site creation invoking `bib2gls` directly.

Note that the test concept based on preferred usage has a considerable weakness: It cannot test warnings, errors and exceptions because they are not preferred usage. On the other hand, it is an important design goal, that the result of this software is trustable if no warning, error or exception occurs. This requires extensive tests also on imperfect runs. These must be supplemented in the future.

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.

[BB24] Javier Bezos and Johannes L. Braams. *Babel User guide*, 1 2024.

[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, `https://ctan.org/pkg/makeidx?lang=en`, 9 2014.

[Car98] David Carlisle. *The longtable package*, v4.09 edition, 5 1998.

[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.

[GNS20] H. Gäßlein, R. Niepraschk, and W. Schmid. *The document class leaflet*, 11 2020.

[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.

[Hob24] John D. Hobby. *MetaPost, a user's manual*, 2 2024. for version 2.10, `https://www.tug.org/docs/metapost/mpman.pdf`.

[Ilt12] Philip Ilten. *The svg Package*, v1.0 edition, 9 2012. philten@cern.ch.

[ISO20] ISO. *Document management – Portable document format – Part 2: PDF 2.0*, 2 edition, 12 2020.

[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.

[Koh23] Markus Kohm. *Die Anleitung KOMA - Script*, 6 2023. Refers to KOMA-script versions 3.36 and 3.37.

[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.

[MF23] F. Mittelbach and U. Fischer. *The documentmetadata-support code*, 3 2023. A copy is within the documentation of this software, in fact two documents, documentmetadata-support-doc.pdf and documentmetadata-support-code.pdf which also comprises the implementation.

[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 transparent package*, v1.1 edition, 5 2016.

[Obe22] Heiko Oberdiek. *The rerunfilecheck package*, v1.10 edition, 7 2022.

[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.

[Rei] E. Reißner. *Manual for the latex-maven-plugin and for an according ant-task, Version X.Y.* The current version is vailable at `http://www.simuline.eu/LatexMavenPlugin/manualLMP.pdf`.

[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.

[Rei23a] E. Reißner. Presentation with/of the latex-maven-plugin. presentation available at `http://www.simuline.eu/LatexMavenPlugin/docClasses/useBeamerPres.pdf`, handout at `http://www.simuline.eu/LatexMavenPlugin/docClasses/useBeamerHandout.pdf`, 10 2023. Comprises both, presentation and handout.

[Rei23b] E. Reißner. Special and common aspects of pdf/dvi/xdvi generators. `http://www.simuline.eu/LatexMavenPlugin/latex/latexEngines.pdf`, 3 2023.

[Rei24a] E. Reißner. Leaflet with/of the latex-maven-plugin. available at `http://www.simuline.eu/LatexMavenPlugin/docClasses/productLeaflet.pdf`, 2 2024.

[Rei24b] E. Reißner. A letter on/with the latex-maven-plugin. available at `http://www.simuline.eu/LatexMavenPlugin/docClasses/letter.pdf`, 2 2024.

[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.

[SGNS20] J. Schlegelmilch, H. Gäßlein, R. Niepraschk, and W. Schmid. *The leaflet document class*, 6 2020.

[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.

[Tal24a] N. L.C. Talbot. The glossaries package v4.54: a guide for beginners. https://www.ctan.org/pkg/glossaries?lang=de, 4 2024.

[Tal24b] N. L.C. Talbot. *User Manual for glossaries.sty v4.54*. dickimaw-books, https://www.ctan.org/pkg/glossaries?lang=de, 4 2024.

[Tan23] T. Tantau. *TikZ and PGF Manual for Version 3.1.10*. Institut für Theoretische Informatik, Universität zu Lübeck, Lübeck, Germany, 1 2023. https://mirror.physik.tu-berlin.de/pub/CTAN/graphics/pgf/base/doc/pgfmanual.pdf.

[Tea00] The LaTeX3Project Team. *LaTeX2ε font selection*, 9 2000.

[Tea20] The Dvipdfmx Project Team. *Dvipdfmx User's Manual*, 6 2020. Version 0.12.4b.

[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.

[TWM23] T. Tantau, J. Wright, and V. Miletić. *The BEAMER class*. joseph.wright@morningstar2.co.uk, 5 2023.

[Ume10] Hideo Umeki. *The geometry package*. latexgeometry@gmail.com, v5.6 edition, 9 2010.

[WK16] Thomas Williams and Colin Kelley. *gnuplot 5.0 – An Interactive Plotting Program*. http://www.gnuplot.info/docs_5.0/gnuplot.pdf, 1 2016. Version 5.0.2.

[WK23] Thomas Williams and Colin Kelley. *gnuplot 6.0 – An Interactive Plotting Program*. http://www.gnuplot.info/docs_6.0/Gnuplot_6.pdf, 12 2023. Version 6.0.

[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

Chapter 14

LaTeX Commands

Glossary

LaTeX engine A compiler for LaTeX files. 13, 189
LaTeX main file A LaTeX file intended to be compiled by a LaTeX engine.. 3, 13, 17, 28

Acronyms

AUX auxiliary file: input and output file for LaTeX engines; read also e.g. by `bibtex`. 86

BBL bibliography for a latex document in latex format: written by the `bibtex` tool and read by LaTeX processors. 89
BCF bibliography content file (?): generated by LaTeX engines if used with package `biblatex`. 86
BST Bibliography Style File read by the `bibtex` tool. 88, 89

DEPYTXC File containing information to replace code snippets in the TEX file by the result of their evaluation; output format of LaTeX engines with package `pythontex` if loaded with option `depythontex`. 87, 142
DOC outdated document format for MS Word. 35
DOCX current document format for MS Word. 13, 14, 35, 83
DPLG `depythontex` log file: home-brewed since the original application does not write log files. 142
DVI DeVice Independent; traditional output format of LaTeX engines, today widely replaced by PDF. 13, 66, 83, 85, 108, 117, 192

EPS Encapsulated PostScript. 16, 34, 66, 67, 74, 118

FIG native file format for xfig. 16, 34, 58, 65, 118
FLS FiLeS dependencies: list of files the according tex file depends on; output format of LaTeX engines if used with option `-recorder`. 67

GIF Graphics Interchange Format, allows also animations. 165
GLG `makeglossaries` log file. 140
GLO GLOssary file containing unsorted and multiple glossary entries; output format of LaTeX engines with package `makeglossaries`. 86
GLS glossary file containing sorted, unified and formatted glossary entries; output format of the `makeglossaries` tool read by LaTeX engines. 86
GP GnuPlot file format. 65

HTML HyperText Markup Language. 13, 14, 17, 66, 83

IDX InDeX file containing unsorted and multiple index entries; output format of LaTeX engines with package `makeindex` or similar. 7, 86, 90–93
IND INDex file containing sorted, unified and formatted index entries, output format of `makeindex` and `xindy`. 86, 90
IST (make-)Index Style File: output format of LaTeX engines if used with package `glossaries` configured for `makeindex`. 86

JPG Graphics format developed by the Joint Photographic Experts Group. 34, 63, 64, 82

MP MetaPost: input format for the graphic program `mpost`. 16, 34, 76
MPS metapost's postscript like output including text. 34, 64, 77
MPX metapost TEX output: texts. 77

ODT Open Document Text. 13, 14, 83
OUT contains bookmarks: input and output format of LaTeX engines if used with package `hyperref`, file ending seems naive. 87

PDF Portable Document Format. 13, 14, 16, 34, 82, 83, 85, 117
PLG `pythontex` log file: home-brewed since the original application does not write log files. 141, 142
PNG Portable Network Graphics. 34, 63, 64, 74, 76, 82, 109, 143, 165
PTX pdf/postscript TeX format; home-brewed. 66, 67
PYTXCODE Code file consisting mainly of code snippets from the TEX file; output format of LaTeX engines with package `pythontex`. 86, 141

SGML Standard Generalized Markup Language. 17
SVG Scalable Vector Graphics. 16, 34, 64, 65, 74, 76, 78, 79

TEX TeX the format, which may also be LaTeX. 86

XDV eXtended Device Independent; an extension of the traditional output format DVI of LaTeX engines, today widely replaced by PDF. 66, 85, 117
XDY index style file for `xindy`: output format of LaTeX engines if used with package `glossaries` configured for `xindy`. 86
XHTML eXtensible HyperText Markup Language. 14
XML eXtensible Markup Language. 17