limap: A LaTeX Package and Class for Typesetting Information Maps

Gerd Neugebauer
Net: gene@gerd-neugebauer.de

This file documents limap.dtx version 2.1 as of 2016/05/29.
Documentation date: 2016/05/29

The Information Mapping® method provides a methodology for structuring and presenting information. It claims to be useful for readers who are more concerned about finding the right information than reading the document as a whole. Thus short, highly structured, and context free pieces of information are used.

limap provides a LaTeX package and a LaTeX class. The package contains definitions to typeset maps and blocks according to the Information Mapping® method. The class provides all definitions to typeset a whole document.

Copyright

Licenses

The source files of `limap` and the derived files

- `limap.dtx`
- `limap.ins`
- `README.md`
- `Makefile`
- `limap.sty`
- `limap.cls`
- `limap.pdf`

may be distributed under the terms of the LaTeX Project Public License version 1.3c, as described in the file `lppl.txt`.

The documentation can be used under the Creative Commons Attributation-Share Alike 4.0 License (CC BY-SA 4.0).

The files in the `samples` directory are distributed unter Creative Commons CC0 1.0 Universal.

Net

The sources of `limap` are hosted on Sourceforge as part of the project `gene-tex-lib`. The sources can be found under the URL `https://sourceforge.net/p/gene-tex-lib/svn/HEAD/tree/limap/`.

A bundled distribution can be obtained via CTAN under the package URL `https://www.ctan.org/pkg/limap` or from the package's home page under `http://www.gerd-neugebauer.de/sowftware/TeX/limap`.

Contact

The author can be contacted under the following coordinates:

Gerd Neugebauer

Im Lerchelsböhl 5
64521 Groß-Gerau
Germany

gene@gerd-neugebauer.de

Contents

Motivation

Methodology

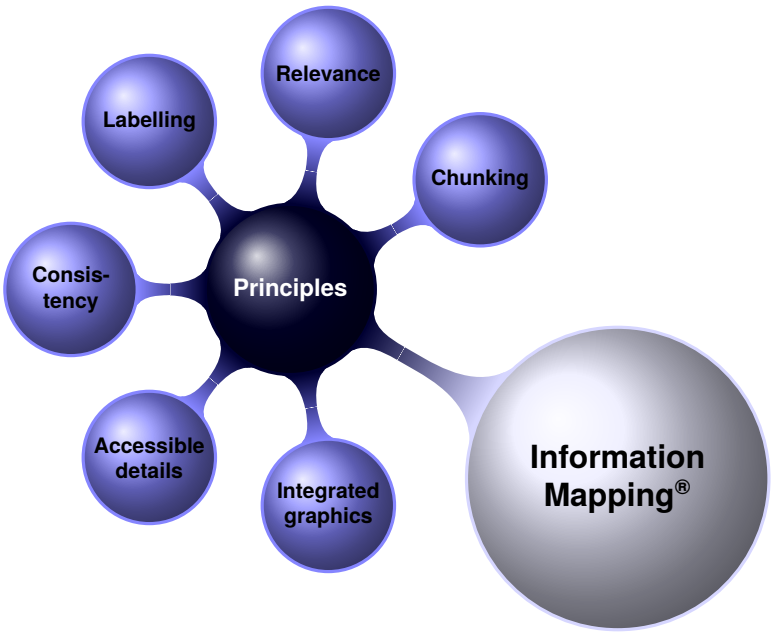
The information mapping® method provides a methodology to structure information in a special way. The aim is to help a reader who uses the document to search for relevant information instead of consuming it from start to end. The information mapping method also claims to raise the productivity of writers.

No tutorial

This document does not include an introduction to the information mapping method itself. The reader is referred to other documents. Maybe an accompanying document will be distributed along with this package.

Unfortunately the methodology – or the name – is protected by a trademark. Thus the teaching of the methodology is restricted to licensed institutions.

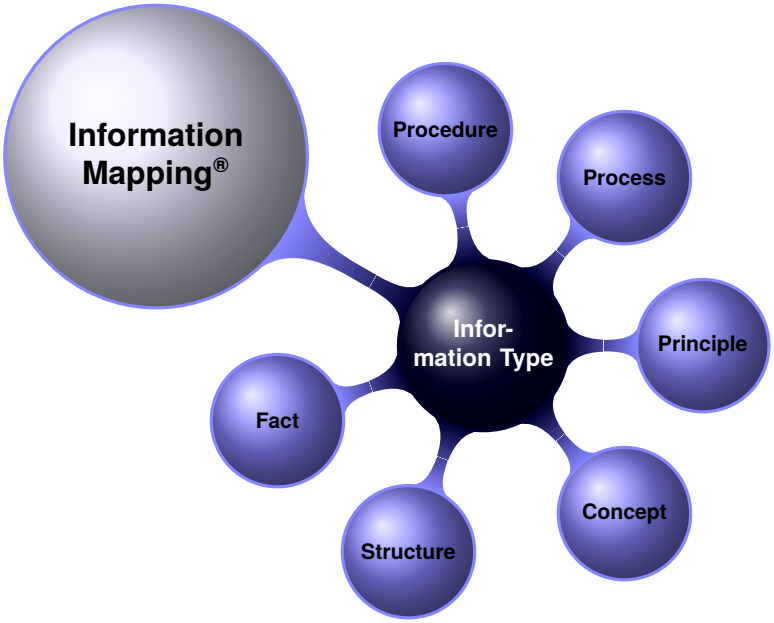
Principles



Continuing…

Motivation, Continued

Information Type



Problems of restructuring

You as LaTeX user may have encountered the problem when restructuring a document: The sections know exactly the level they belong to. If you introduce a new section and put another section into it you have to change all sectioning commands to reflect the new level of the section and their subsections.

The macros of `limap` abstract away the level of the document structuring. Any structuring unit is a “Block”. It may contain text or other blocks. Such blocks are the replacement or the sectioning commands. When you restructure your document you simply shift the block and anything it contains to the new place and you are done.

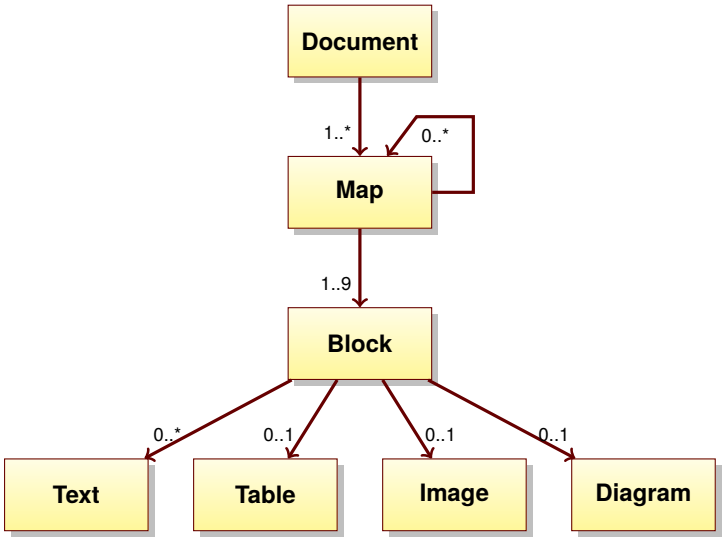
The same mechanism can help you to include the same material in several documents – at different sectioning levels!

Continuing…

Motivation, Continued

Structure and constituents

The general approach focuses very much on the structure of the document. The following diagram illustrates the terminology used within this documentation.



Class or package

To support the information mapping® method several LaTeX macros and environments are provided which allow you to enter a logical description of the relevant concepts. Those macros are provided in the package and class file. It is up to you to choose one of them.

Interoperabilty with other classes and packages

The main part of the user interface is inherited from LaTeX. The major differences are the sectioning commands which are made obsolete in parts by the information mapping® method. Thus most packages can be used to typeset contents of a block.

Getting Started

Introduction

If you are starting to use `limap` you should be vaguely familiar with the underlying methodology. This documentation does not provide an introduction.

When you start using this class or package you should say “Good bye” to the classical document structuring macros in the LaTeX standard classes.

Class or package

First of all you should decide whether to use the `limap` class or the `limap` package. The class is meant for a complete document. The package allows to combine `limap` with arbitrary other document classes.

Contents

In the maps contained herein you will find an introduction on the use of the `limap` macros and environments. They are accompanied by illustrating sample code.

Using the Document Class

Using the document class

This package provides both a class file as well as a package. The package contains the definitions of maps, blocks, and others. They can be used together with any base class. This is illustrated in the following preamble:

```
\documentclass{book}
\usepackage{limap}
\begin{document}
```

Suppressing the block lines

This class option `nolines` can be used to suppress the line above and below block. This is illustrated in the following preamble:

```
\documentclass[nolines]{limap}
\begin{document}
```

Other options

Any option not processed by `limap.cls` is passed to the underlying document class used. Thus it is possible to customize the underlying class any further.

Several types of options

In the maps contained herein you will find the supported options for the document class `limap`.

Document Class Options for Language Selection

Introduction

The document class `limap` inserts some words. Thus it has to know the language it is supposed to use. Thus the language has to be specified as one document class option.

```
\documentclass[german]{limap}
\begin{document}
```

Supported languages

First, we describe the settings influencing the language specific settings. They do not make provisions to use the appropriate hyphenation patterns. They just arrange things such that the internally used texts are displayed in the chosen language.

Option	Description
`austrian`	Activate the language specific text fragments for the Austrian language (in fact German with one minor modification).
`english`	Activate the language specific text fragments for the English language.
`french`	Activate the language specific text fragments for the French language.
`german`	Activate the language specific text fragments for the German language.
`USenglish`	Activate the language specific text fragments for the American English.

Default language

The default language is `english`. It is used if no language is set.

Beware of hyphenation

Note that the hyphenation patterns are not loaded automatically. You have to load the hyphenation patterns, for instance with the babel package.

This behavior has been chosen to allow you to select the language package of your choice. Also you can pass in additional options to this package more easily.

Document Class Options for Variant Selection

Introduction

The document class `limap` is based on another document class. Thus you can use the well-known macros and environments defined there and take advantage of the extensions provided by `limap`.

```
\documentclass{limap}
\begin{document}
```

Variants

The class has two additional options to determine the base class to be used. The first option is the variant. It can take the following values:

Option	Description
`base`	Use the base set of classes. This is the default.
`koma`	Use the set of classes from koma-script.

Default variant

If no document class option is used then `base` is the default variant.

Document Class Options for Base Class Selection

Introduction

The document class `limap` follows the logic introduced in the LaTeX standard document classes. The type of the base document class can be given as argument to the `limap` class.

```
\documentclass[book]{limap}
\begin{document}
```

Class type

The second option is the class type. It determines which kind of document to typeset. It can take the following values:

Option	Description
`book`	Typeset a book type document.
`report`	Typeset a report type document.
`article`	Typeset an article type document.
`letter`	Typeset a letter type document.

Mapping of variant and type

The following table shows which base classes are loaded according to the given values:

Type /variant	*base*	*koma*
`book`	book	scrbook
`report`	report	scrreprt
`article`	article	scrartcl
`letter`	letter	scrlettr

Default type

The default type is `report`. It is used if no type is set.

Using the Package

Introduction

The package `limap` can be used together with most document classes. An exception are classes for slides like the beamer class.

As usual it is declared in the document preamble.

```
\documentclass{report}
\usepackage{limap}
\begin{document}
```

Content

The following additional information is available.

Package Options for Language Selection

Introduction

The package `limap` inserts some words. Thus it has to know the language it is supposed to use. Thus the language has to be specified as package option.

```
\usepackage[german]{limap}
\begin{document}
```

Inheritance of document class options

Alternatively the language selecting option can be specified as document class option. Thus several packages can share the same setting.

```
\documentclass[german]{scrbook}
\documentclass{limap}
\begin{document}
```

Supported languages

Here we describe the settings influencing the language specific settings. They do not make provisions to use the appropriate hyphenation patterns. They just arrange things such that the internally used texts are displayed in the chosen language.

Option	Description
`austrian`	Activate the language specific text fragments for the Austrian language (in fact German with one minor modification).
`english`	Activate the language specific text fragments for the English language.
`french`	Activate the language specific text fragments for the French language.
`german`	Activate the language specific text fragments for the German language.
`USenglish`	Activate the language specific text fragments for the American English.

Default language

The default language is `english`. It is used if no language is set.

Continuing…

Package Options for Language Selection, Continued

Beware of hyphenation

Note that the hyphenation patterns are not loaded automatically. You have to load the hyphenation patterns, for instance with the babel package.

This behavior has been chosen to allow you to select the language package of your choice. Also you can pass in additional options to this package more easily.

The Block

Meaning

A “block” is the essential building units of the `limap` package. You can think of it as paragraph with a title.

Appearance

The blocks are usually typeset with the block label on the left side and the contents to its right. They are surrounded by white-space and a thin line above and below.

`\Block`

The macro `\Block` can be used to typeset an block. It takes one argument which is the block label.

This is a shorthand for denoting a block. The end mark can be omitted if you use the macro instead of the environment. Nevertheless this is depreciated.

Example

The following sample illustrates how to enter a block.

```
\Block{Block Label}
And now comes the block text. It can consist of one or more
classical paragraphs, or tables, or pictures, or something
else.
```

Just in a map

Note that the macro `\Block` can be used inside a map only. If you try to use it outside the scope of a map you will get an error.

More on blocks

Several more aspects of blocks are covered in the following maps.

Configuring Blocks

`\MapBlockLabelFont`

The macro `\MapBlockLabelFont` determines the font changing command to be used for typesetting the block label. The default is empty.

Colored block labels

The macro `\MapBlockLabelFont` can for instance be used to achieve colored block labels. For this purpose we can include the package `xcolor` in the preamble and select for instance a named color for the rule. This is illustrated in the following example.

```
\usepackage[svgnames]{xcolor}
\renewcommand\MapBlockLabelFont{\color{Navy}}
```

`\MapParskip`

The macro `\MapParskip` determines the vertical distance of the text from the separating rules. The default is 2ex.

```
\renewcommand\MapParskip{.25ex}
```

`\MapTitleFraction`

The macro `\MapTitleFraction` determines the part of the page width devoted to the block label area. It is a fraction in the range from 0 to 1. The default value of `\MapTitleFraction` is 0.2.

```
\renewcommand\MapTitleFraction{.25}
```

`\MapTextFraction`

This macro determines the part of the page width devoted to the text area. It is a fraction in the range from 0 to 1. The default value of `\MapTextFraction` is 0.75.

`\MapTitleFraction` and `\MapTextFraction` should add up to something less or equal to 1. Otherwise you will get some “overfull hbox” messages.

```
\renewcommand\MapTextFraction{.8}
```

Configuring the Rules of Blocks

Motivation

Blocks are usually surrounded by horizontal rules. The appearance of these rules can be influenced by some macros.

`\MapRuleWidth`

The macro `\MapRuleWidth` determines the width of the rules drawn between blocks. It is defined as a macro containig a length. The default is 1pt.

```
\setlength\MapRuleWidth{1mm}
```

Suppress visible rules

For some people the rules are distracting since they emphasize the structure too much. In such situations the rule width might be reduced. In the extreme case it can even be set to 0pt to suppress the lines at all as in the following example:

```
\renewcommand\MapRuleWidth{0pt}
```

`\MapRuleStart`

The macro `\MapRuleStart` is inserted before rules around blocks. It can be used to inject some code before the rule is typeset. Initially it is empty.

Colored rules

The macro `\MapRuleStart` can for instance be used to achieve colored rules. For this purpose we can include the package `xcolor` in the preamble and select a named color for the rule. This is illustrated in the following example.

```
\usepackage[svgnames]{xcolor}
\renewcommand\MapRuleStart{\color{Silver}}
```

A Wide Block

Motivation

Sometimes it is necessary to extend the content of a block to the whole width of the page. This can be the case for illustrations or large tables.

`\WideBlock`

The macro `\WideBlock` can be used to typeset a piece of information on the whole page width. It is normally used after an initiating block containing the title of the whole construction.

The macro `\WideBlock` takes one argument which contains the material to span the whole page width.

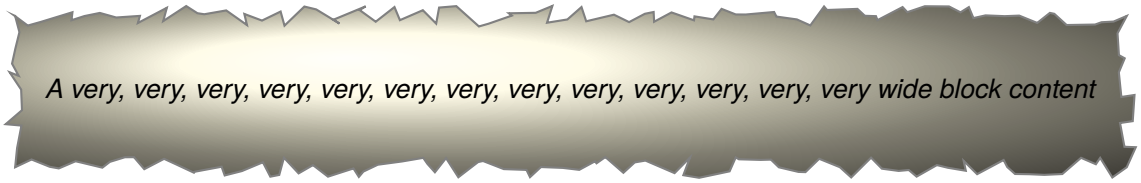
Example

The following sample illustrates how to enter a wide block.

```
\Block{Block Label}
And now comes the block text.
\WideBlock{\includegraphics{images/overview.svg}}
```

Block Label

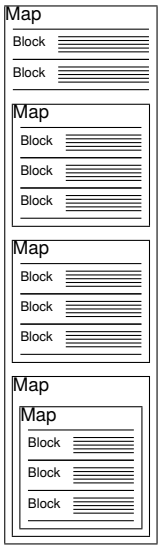
And now comes the wide block produced with the help of Ti*k*Z and its decoration with random steps.



The Map

Structuring documents

The maps are the structuring units of the `limap` package. Maps may recursively contain maps and other material. The other material is usually named a block. This recursive structure is illustrated in the following figure.



`Map`

The environment `Map` can be used to typeset a map. It takes a single argument which contains the map title. The map title is typeset above the map and is repeated on each continuation page.

Example

```
\begin{Map}{Map Title}
  \Block{Block Label}
   And now comes the block text. It can  consist of
  one or more classical paragraphs, or tables, or
  pictures, or something else.
\end{Map}
```

Continuing…

The Map, Continued

To at most 12 cascaded Maps

There is a technical restriction in the current implementation of Maps. This restriction does not allow more than 12 cascaded Maps, i.e. Maps in Maps …in Maps.

According to the good old 7±2 rule a full populated document of this level would contain 282 429 536 481 Maps. I think it should take some time until this limit is reached.

More on maps

Some more aspects are covered in the following pages.

Referencing Maps

Referencing maps

Maps can be referenced in the usual LaTeX way. For this purpose you place a macro invocation of `\label` right behind the beginning of the `Map`.

Then you can add references somewhere in the document with `\ref` and `\pageref`. For backward references two LaTeX runs are required.

Example

```
\begin{Map}{Map Title}\label{my.label}
  \Block{Block Label}
  As said on page~\pageref{my.label}...
```

Caveat emptor

`limap` automatically labels any map with a label of the form `Map@`*n* where *n* is a sequence number. Thus avoid to use such labels yourself.

Configuring Maps

`\MapFont`

The macro `\MapFont` determines the font changing command to be used when starting a new map.

```
\renewcommand\MapFont{\tt}
```

`\MapTitleFont`

The macro `\MapTitleFont` determines the font changing command to be used when typesetting the title of a map. The default is `\Large`.

```
\renewcommand\MapTitleFont{\huge\bfseries}
```

`\MapContinued`

The macro `\MapContinued` contains the text appearing at the end of map which are continued on the next page. It is initiated when the class or package is loaded. It can overwritten afterwards – for instance in the preamble.

```
\renewcommand\MapContinued{}
```

`\MapContinuing`

The macro `\MapContinuing` contains the text appearing at the beginning of map which are continued from the previous page. It is typeset after the map title. It is initiated when the class or package is loaded. It can overwritten afterwards – for instance in the preamble.

```
\renewcommand\MapContinuing{}
```

`\MapTitleContinued-Font`

This macro determines the font changing command to be used for typesetting the additional text after titles on followup pages of multi-page maps.

The default value is `\small`.

```
\renewcommand\MapTitleContinuedFont{\normalsize}
```

Continuing…

Configuring Maps, Continued

`\MapNewpage`

The macro `\MapNewpage` is expanded whenever a new page is required between maps. Thus it can be used to suppress the newpages by `\let`ing it to `\relax`. Note that this is not in the spirit of the Information Mapping® method.

`\MapTOC`

The macro `\MapTOC` is expanded to generate the entry in the table of contents. It can be redefined to allow another behavior.

The Table of Contents

Everything is local

The concept of the underlying methodology is that everything should be addressed relative to the current location. We see this when dealing with maps (see page 19). The same principle is applied to the table of contents.

Direct children are included

The table of contents includes all maps contained in the map in which it appears. This means the immediate children of the map are shown.

Block context required

The table of contents is typeset inside a block (see page 15).

`\MapTableOfContents`

The macro `\MapTableOfContents` can be used to typeset the table of contents for a map. This table of contents includes all sub-maps of the map it is contained in – not recursively but only one level deeper.

The macro `\MapTableOfContents` does not take any argument.

Example

```
\Block{Contents}
And some wise words about the table of contents.
\MapTableOfContents
```

`\MapTableOfContentsStyle`

The macro `\MapTableOfContentsStyle` can be used to determine the style of the `\MapTableOfContents` The default style is an open layout utilizing the package booktabs.

The macro `\MapTableOfContentsStyle` takes one argument. This argument may have one of the following values:

Argument	Meaning
open	an open style for the TOC
boxed	a boxed style for the TOC

Tables

Introduction

Tables play an important role as part of maps. Usually they are included into a block and preceded by some useful introduction.

Combination with booktabs

The document class `limap` and the package `limap` automatically include the package booktabs. This package contains some support for typesetting proper tables. You should have a look at the documentation of `booktabs` and follow the recommendations given there.

`booktabs` is automatically loaded by `limap` upon start-up.

`MapTabular`

The environment `MapTabular` provides a convenient way to include a table into a block. It produces a `tabular*` environment and sets the width to the width of the text column of the block.

Example

```
\Block{Table block}
Sheding some insight on the following table.

\begin{MapTabular}{lll}\toprule
  a & b & \\\midrule
  x & y & Z\\\hline
  X & Y & Z\\\bottomrule
\end{MapTabular}
```

`\MapTabularFraction`

The tabular is always centered in the text column of the block. The macro `\MapTabularFraction` contains a factor for the line width occupied by the tabular. This value is used to determine the width of the tabular. The default value for the macro `\MapTabularFraction` is 0.95.

```
\renewcommand\MapTabularFraction{.7}
```

Configuration

Introduction

`limap` is designed with a set of extension points. Those are mainly definition which can be overwritten to achieve a certain effect.

Configuration in the preamble

Any configuration change performed in the preamble of a document is global. It lasts until oberwritten within the document.

Configuration in the document

The configuration parameters can be overwritten in the document just before they are needed. This is not recommended since it makes it harder to achieve consistency.

External configuration

The confiration can be externalised. For this purpose a special file `limap.cfg` is loaded with the class or package in case it exists. This file can contain configuration options for `limap`. Thus it is possible to share the same appearance among different documents.

Contents

The following maps contain the various aspects of the configuration of `limap`.

The Configuration File limap.cfg

Default configuration file

When the class or package is loaded as a last action a configuration file is loaded if it can be found. The name of the configuration file is `limap.cfg`. This file can contain re-definitions of the several macros to adjust the behavior of `limap` on a per directory, per user or per installation base.

Note

Some settings are activated before the configuration file is loaded. Thus some settings may not have any effect at all.

Changing or Adding Language Specific Settings

Overview

Several strings are used automatically by the current class or package. Default values for several languages are hardwired in the implementation. Nevertheless it is possible to change those language specific settings.

If you create settings for a new language it is highly recommended to contact the author to integrate them into the default distribution.

The following macros can be redefined in the preamble after the package or class has been loaded to reset the language specific text.

`\MapTOCname`

The macro `\MapTOCname` contains the text of the heading in table of contents of maps for the column of map titles. It is initiated when the class or package is loaded. It can overwritten afterwards – for instance in the preamble.

```
\renewcommand\MapTOCname{Issue}
```

`\MapTOCpage`

The macro `\MapTOCpage` contains the text of the heading in table of contents of maps for the column of page numbers. It is initiated when the class or package is loaded. It can overwritten afterwards – for instance in the preamble.

```
\renewcommand\MapTOCname{Reference}
```

Providing a new language

If you want to provide a new language *lang* you can define the macro `\LIMAP@SelectLanguage@`*lang* which redefines the macros given above. This definition has to be present before the package is loaded.

Note that the macro name contains the @ character. Thus the definition should be made in a package of its own.

The Implementation

Overview

This part of the document describes the implementation. Usually it is not meant for the casual user. Nevertheless it might be fruitful for those searching for inspiration or for tricks when using this class or package.

Contents

The Version Information

Purpose

The version information is included for printing it on the documentation and at start-up when the class or package is loaded. It has to preceed the documentation driver to properly include this information into the printed manual.

`\filename` `\filename` is the name of the dtx file containing this class and package.

```
1 \def\filename{limap.dtx}
```

`\fileversion` `\fileversion` is the version number of the `dtx` file. It is used as a version number for the class and package.

```
2 \def\fileversion{2.1}
```

`\filedate` `\filedate` is the change date of the `dtx` file. It is used as a version date and documentation date.

```
3 \def\filedate{2016/05/29}
```

`\docversion` `\docversion` is the version number of the documentation. It is identical to the version number of the `dtx` file.

```
4 \let\docversion=\fileversion
```

`\docdate` `\docdate` is the change date of the documentation. It is identical to the change date of the `dtx` file.

```
5 \let\docdate=\filedate
```

The Documentation Driver

Purpose

The documentation driver is necessary to provide a self documenting `dtx` file. With this construction the dtx file can be run through LaTeX to produce the documentation.

The driver section contains a complete LaTeX document which loads the `dtx` file. The special class `ltxdoc` is used and some arrangements are made for this purpose.

Driver code

The driver code is not exported by the installer.

```
⟨*driver⟩
\documentclass[a4paper]{ltxdoc}
\RequirePackage{textcomp}
\usepackage{limap}
\let\LimapFilename\filename
\let\LimapFileversion\fileversion
\let\LimapFiledate\filedate
\let\LimapDocdate\docdate
```

Page Layout

```
\oddsidemargin=10pt
\evensidemargin=10pt
\textwidth=430pt
\textheight=650pt
\voffset=-12mm
```

Headings

```
\usepackage{fancyhdr}
\addtolength{\headheight}{2ex}%
\pagestyle{fancy}%
\cfoot{}
\rhead{\small\sf\thepage}
\lhead{\textit{\footnotesize limap}  Package and Class}
```

Links and such

```
\usepackage{hyperref}
```

Continuing…

The Documentation Driver, Continued

Fonts

```
\usepackage{fontspec}
\setmainfont{TeX Gyre Heros}
\setsansfont{TeX Gyre Heros}%[Scale=MatchLowercase]
\setmonofont{Inconsolata}%[Scale=MatchLowercase]
```

Coloring

```
\usepackage[svgnames]{xcolor}
\renewcommand\MapTitleFont{\Large\bfseries\color{Navy}}
\renewcommand\MapBlockLabelFont{\bfseries\color{Navy}}
\definecolor{linkColor}{rgb}{.66,.2,.2}
\hypersetup{colorlinks,
            citecolor=linkColor,
            filecolor=linkColor,
            linkcolor=linkColor,
            urlcolor=linkColor}
\renewcommand\MapRuleStart{\color{Navy}}
\renewcommand\MapRuleWidth{1pt}
```

Graphics

```
\usepackage{tikz}
\usetikzlibrary{decorations.pathmorphing}
\usetikzlibrary{shadows}
\usetikzlibrary{mindmap}
\usetikzlibrary{calc}
```

Saving version

```
\let\filename\LimapFilename
\let\fileversion\LimapFileversion
\let\filedate\LimapFiledate
\let\docdate\LimapDocdate
```

Adapting doc.sty

```
\def\theCodelineNo{\color{DarkGreen}\rmfamily\scriptsize\arabic{CodelineNo}}%
\makeatletter
\renewcommand\DescribeMacro{\catcode`\\=12\Describe@@Macro}
\def\Describe@@Macro#1{\Block{\PrintDescribeMacro{#1}}%
  \SpecialUsageIndex{#1}\@esphack\ignorespaces}
\renewcommand\DescribeEnv{\catcode`\\=12\Describe@@Env}
\def\Describe@@Env#1{\Block{\PrintDescribeEnv{#1}}%
  \SpecialEnvIndex{#1}\@esphack\ignorespaces}
\makeatother
\let\maketitle\MakeTitle
```

Continuing…

The Documentation Driver, Continued

Some additions

```
\newcommand\R{\(^{\textrm{\footnotesize\textregistered}}\) }
\let\marginpar\Block
\InputIfFileExists{limap.dcf}{}{}
\RecordChanges
\EnableCrossrefs
\CodelineIndex
```

The content

Now everything is prepared. Let the show begin…

```
\begin{document}
\DeleteShortVerb{|}
\DocInput{\filename}
%\newpage
%\PrintChanges
\newpage
\setcounter{IndexColumns}{2}
\PrintIndex
\end{document}
⟨/driver⟩
```

The TEX Code

Overview

The rest of the document describes the implementation. Usually it is not meant for the casual user. Nevertheless it might be fruitful for those searching for inspiration or for tricks when using this class or package.

Contents

The Package and Class Declarations

Preliminaries

First of all we request a descent version of LaTeX to be used. I don't think ikt does not have to be too new.

```
\NeedsTeXFormat{LaTeX2e}
```

Package identification

When the package is generated, the package identification is included.

```
⟨*package⟩
\ProvidesPackage{limap}[\filedate\space Gerd Neugebauer]
⟨/package⟩
```

Class identification

When the class is generated, the class identification is included.

```
⟨*class⟩
\ProvidesClass{limap}[\filedate\space Gerd Neugebauer]
⟨/class⟩
```

Language Specific Macros

Introduction

This section contains internal macros used to implement the functionality. New languages can be easily be added. For this purpose only a new macro has to be defined and a package/class option for the convenience of the user.

Consider you want to add a new language “latin” then you have to provide the command `\LIMAP@SelectLanguage@latin`. This macro should simply redefine the macros containing strings of the language specific texts. Examples for other languages are provided in this section.

To enable the language settings for “latin” the macro `\LIMAP@Language` has to be defined to contain the value “latin”. Usually this is accomplished by providing a convenient option to the package or class.

`\defineLimapLanguage` Provide the definitions for a language. The different texts to be used are stored in a macro which defines the target macros when expanded.

```
\def\defineLimapLanguage#1#2#3#4#5{%
  \expandafter\def\csname LIMAP@SelectLanguage@#1\endcsname{%
    \def\MapContinued{#2}%
    \def\MapContinuing{#3}%
    \def\MapTOCname{#4}%
    \def\MapTOCpage{#5}%
  }%
}
```

Definitions for the language “austrian“

```
\defineLimapLanguage{austrian}%
  { Fortsetzung}{Fortsetzung\dots}%
  {Titel}{Seite}
```

Definitions for the language “german“

```
\defineLimapLanguage{german}%
  { Fortsetzung}{Fortsetzung\dots}%
  {Titel}{Seite}
```

Definitions for the language “english“

```
\defineLimapLanguage{english}%
  { Continued}{Continuing\dots}%
  {Title}{Page}
```

Continuing…

Language Specific Macros, Continued

Definitions for the language "USenglish"

```
\defineLimapLanguage{USenglish}%
  { Continued}{Continuing\dots}%
  {Title}{Page}
```

Definitions for the language "french"

```
\defineLimapLanguage{french}%
  { continuation}{continuation\dots}%
  {Intitulé}{Page}
```

`\LIMAP@Language` The macro `\LIMAP@Language` determines the language to be used for several small text fragments to be inserted at certain places. It is redefined by package/class options and evaluated at the end to activate the selected settings.

```
\providecommand\LIMAP@Language{english}
```

```
\DeclareOption{austrian}{\renewcommand\LIMAP@Language{austrian}}
\DeclareOption{german}{\renewcommand\LIMAP@Language{german}}
\DeclareOption{french}{\renewcommand\LIMAP@Language{french}}
\DeclareOption{english}{\renewcommand\LIMAP@Language{english}}
\DeclareOption{USenglish}{\renewcommand\LIMAP@Language{USenglish}}
```

`\ifLIMAP@strict` The boolean `\ifLIMAP@strict` determines if the lower sectioning macros should be disabled in the class.

```
\newif\ifLIMAP@strict \LIMAP@stricttrue
```

```
\DeclareOption{nonstrict}{\LIMAP@strictfalse}
```

```
\DeclareOption{nolines}{\def\MapRuleWidth{0pt}\ignorespaces}
```

Determining the Appropriate Base Class

```
115 ⟨*class⟩
```

`\LIMAP@ClassType` The macro `\LIMAP@ClassType` determines the type of the class to be used. Usually it can take the values `book`, `report`, `article`, and `letter` (for completeness). This macro is redefined when the options of the class are evaluated. Finally this macro helps to select the appropriate base class.

```
116 \providecommand\LIMAP@ClassType{report}
```

```
117 \DeclareOption{book}{\renewcommand\LIMAP@ClassType{book}}
118 \DeclareOption{report}{\renewcommand\LIMAP@ClassType{report}}
119 \DeclareOption{article}{\renewcommand\LIMAP@ClassType{article}}
120 \DeclareOption{letter}{\renewcommand\LIMAP@ClassType{letter}}
```

`\LIMAP@Variant` The macro `\LIMAP@Variant` determines the variant of the class to be used. Usually it can take the values `base` and `koma`. This macro is redefined when the options of the class are evaluated. Finally this macro helps to select the appropriate base class.

```
121 \providecommand\LIMAP@Variant{base}
```

Options for selecting the variant

```
122 \DeclareOption{koma}{\renewcommand\LIMAP@Variant{koma}}
123 \DeclareOption{base}{\renewcommand\LIMAP@Variant{base}}
```

Mapping to document class

Define a mapping between the variant and class type to the class name to be used.

```
124 \newcommand\LIMAP@Class@base@article{article}
125 \newcommand\LIMAP@Class@base@report{report}
126 \newcommand\LIMAP@Class@base@book{book}
127 \newcommand\LIMAP@Class@base@letter{letter}
128 \newcommand\LIMAP@Class@koma@article{scrartcl}
129 \newcommand\LIMAP@Class@koma@report{scrreprt}
130 \newcommand\LIMAP@Class@koma@book{scrbook}
131 \newcommand\LIMAP@Class@koma@letter{scrlettr}
132 ⟨/class⟩
```

Continuing…

Determining the Appropriate Base Class, Continued

Pass on the unknown options

```
133 ⟨*class⟩
134 \DeclareOption*{\PassOptionsToClass{\CurrentOption}{%
135     \csname LIMAP@Class@\LIMAP@Variant @\LIMAP@ClassType\endcsname}%
136   }
137 ⟨/class⟩
```

Thus the class specific options are completed.

Now we can process all options.

```
138 \ProcessOptions
```

```
139 ⟨*class⟩
```

The requested class is loaded and the options remaining are processed.

```
140 \LoadClass{\csname
141   LIMAP@Class@\LIMAP@Variant @\LIMAP@ClassType\endcsname}
142 ⟨/class⟩
```

Loading Required Packages

longtable for breakable tables

The package longtable is used internally to implement a part of the required functionality. Thus we need to ensure that it is loaded.

```
143 \RequirePackage{longtable}
```

etoolbox

```
144 \RequirePackage{etoolbox}
```

booktabs for nice tables

The package booktabs is used internally to implement a part of the required functionality. Thus we need to ensure that it is loaded.

```
145 \RequirePackage{booktabs}
```

fancyhdr for head and foot lines

```
146 ⟨*class⟩
147 \RequirePackage{fancyhdr}
148 \addtolength{\headheight}{2ex}%
149 \pagestyle{fancy}%
150 \cfoot{}
151 \rhead{\small\thepage}
152 \lhead{\textit{\footnotesize\@title}}
153 \def\@title{}
154 ⟨/class⟩
```

No vertical adjustment of pages

Since the blocks are not supposed to line up at the end of the page we declare `\raggedbottom`.

```
155 \raggedbottom
```

Layout Parameters

Overview

The layout can be influenced by a large number of parameters. Thus the design decisions have been made transparent (to a certain degree at least). These options are not meant to be changed except when a new layout is being designed and implemented.

`\MapRuleWidth` The macro `\MapRuleWidth` determines the width of the rules drawn between blocks.

```
156 \providecommand\MapRuleWidth{.25pt}
```

`\MapRuleStart` The macro `\MapRuleStart` is inserted before the rules drawn between blocks.

```
157 \newcommand\MapRuleStart{}
```

`\MapContinued` This macro determines the text to be used in the title of continued maps. This macro is reset when the language specific initializations are performed.

```
158 \newcommand\MapContinued{}
```

`\MapContinuing` The macro `\MapContinuing` determines the text to be used at the bottom of the map which is continued. This macro is reset when the language specific initializations are performed.

```
159 \newcommand\MapContinuing{}
```

Continuing…

Layout Parameters, Continued

`\MapContinuingFormat` This macro determines the format of the bottom line on continued maps. I.e. it includes the text as well as font changing commands. The text is passed to this command as argument 1.

```
160 \newcommand\MapContinuingFormat[1]{\textit{\footnotesize #1}}
```

`\MapContinuedFormat` This macro determines the format of the bottom line on continued maps. I.e. it includes the text passed to it as argument 1 as well as font changing commands.

```
161 \newcommand\MapContinuedFormat[1]{, {\MapTitleContinuedFont #1}}
```

`\MapFont` The macro `\MapFont` determines the font changing command to be used when starting a new map.

```
162 \let\MapFont\textsf
```

`\MapTitleFont` The macro `\MapTitleFont` determines the size changing command to be used when typesetting the title of a map.

```
163 \let\MapTitleFont\Large
```

`\MapTitleContinuedFont` This macro determines the font changing command to be used for typesetting the additional text after titles on followup pages of multipage maps.

```
164 \let\MapTitleContinuedFont\small
```

`\MapBlockLabelFont` This macro determines the font changing command to be used for typesetting the block label.

```
165 \def\MapBlockLabelFont{}
```

Continuing…

Layout Parameters, Continued

`\MapParskip` The macro `\MapParskip` determines the distance of the text from the separating rules.

```
166 \newcommand\MapParskip{2ex}
```

`\MapTitleFraction` The macro `\MapTitleFraction` determines the part of the page width devoted to the title area. It is a fraction in the range from 0 to 1.

```
167 \newcommand\MapTitleFraction{.2}
```

`\MapTextFraction` This macro determines the part of the page width devoted to the text area. It is a fraction in the range from 0 to 1. `\MapTitleFraction` and `\MapTextFraction` should add up to something less or equal to 1. Otherwise you will get some “overfull hbox” messages.

```
168 \newcommand\MapTextFraction{.75}
```

Adaptable Macros

Overview

`\MapNewpage` The macro `\MapNewpage` is expanded whenever a new page is required between maps. Thus it can be used to suppress the newpages by `\let`ing it to `\relax`.

```
\let\MapNewpage\newpage
```

`\MapTOC` The macro `\MapTOC` is expanded to generate the entry in the table of contents. It can be redefined to allow another behavior.

```
\newcommand\MapTOC[1]{%
  \refstepcounter{\@nameuse{Map@TOC@name\the\Map@level}}%
  \addcontentsline{toc}{\@nameuse{Map@TOC@name\the\Map@level}}{#1}%
}
```

`\MapTOCname` The macro `\MapTOCname` contains the heading for the section title in contents blocks. This macro is reset when the language specific initializations are performed.

```
\newcommand\MapTOCname{}
```

`\MapTOCpage` The macro `\MapTOCpage` contains the heading for the page number in contents blocks. This macro is reset when the language specific initializations are performed.

```
\newcommand\MapTOCpage{}
```

`\MapTOCheadfont` The macro `\MapTOCheadfont` contains the font switching command for typesetting the head line of map table of contents.

```
\newcommand\MapTOCheadfont{\scriptsize\emph}
```

Internal Macros, Lengths, and Counters

This section contains internal macros used to implement the functionality.

`\Map@length` The length register `\Map@length` is allocated to store the width of the space between the columns of a block.

```
\newlength{\Map@length}
```

`\Map@level` The macro `\Map@level` determines the level of inclusion of maps. It is used to determine the appearance in the table of contents.

```
\newcount\Map@level
\Map@level=0
```

`\Map@blockcount` The macro `\Map@blockcount` is used to count the blocks per map to issue a package warning if required.

```
\newcount\Map@blockcount
```

`\LT@final@warn` The macro `\LT@final@warn` is defined in `longtable`. It is redefined to show `limap` are originator.

```
\def\LT@final@warn{%
  \AtEndDocument{%
    \PackageWarning{limap}%
    {Table \@width s have changed. Rerun LaTeX.\@gobbletwo}}%
  \global\let\LT@final@warn\relax}
```

Typesetting a Map

`Map` The environment `Map`determines the appearance of a map. It is implemented as a `longtable` environment which takes care for the page breaks and inserts material at the end of the page and the beginning of the new page upon page break.

```
\def\Map#1{%
```

First the messages of `longtable` are modified to show this package name instead.

```
  \def\LT@err{\PackageError{limap}}%
  \def\LT@warn{\PackageWarning{limap}}%
```

The map local macro `\Block` is activated. The counter for blocks is reset.

```
  \let\Block\Map@Block
  \let\endBlock\Map@endBlock
  \Map@blockcount=0
```

The number of the map in the internal counting is set by incrementing the old value.

```
  \global\advance\Map@no1
```

```
  \ifx\Map@UP\empty\else
    \immediate\write\@auxout
      {\string\expandafter\string\xdef\string\csname\space
        Map@parts@\Map@UP\string\endcsname{\string\csname\space
          Map@parts@\Map@UP\string\endcsname\the\Map@no:}}%
  \fi
```

Continuing…

Typesetting a Map, Continued

```
\edef\Map@UP{\the\Map@no}%
\ifnum\Map@level>0
  \xdef\Map@@up{\Map@UP}% Just to save the value across blocks.
  \endgroup
  \Map@end
  \begingroup
  \edef\Map@UP{\Map@@up}%
  \def\@currenvir{Map}%
\fi
\edef\Map@this{\the\Map@no}%
```

The entries for future use of sub-maps are written to the aux file.

```
\immediate\write\@auxout
  {\string\global\string\@namedef{Map@parts@\the\Map@no}{}}%
```

```
\global\advance\Map@level1
\def\Map@TITLE{#1}%
\Map@start
}
```

```
\def\endMap{%
  \Map@end
  \global\advance\Map@level-1
  \ignorespaces
}
```

`\ifMap@open@` The conditional `\ifMap@open@` is used to record the opening and closing of the `longtable` environment, since can not be used inside itself. Thus it can be closed before a new instance is opened.

```
\newif\ifMap@open@
\Map@open@false
```

Continuing…

Typesetting a Map, Continued

`\Map@TOC@name` The macros `\Map@TOC@name`…provide a mapping between a number and a sectioning unit. This mapping is used when the entry in the table of contents is generated.

```
\@namedef{Map@TOC@name0}{chapter}
\@namedef{Map@TOC@name1}{section}
\@namedef{Map@TOC@name2}{subsection}
\@namedef{Map@TOC@name3}{subsubsection}
\@namedef{Map@TOC@name4}{paragraph}
\@namedef{Map@TOC@name5}{subparagraph}
\@namedef{Map@TOC@name6}{subsubparagraph}
\@namedef{Map@TOC@name7}{subsubparagraph}
\@namedef{Map@TOC@name8}{subsubparagraph}
\@namedef{Map@TOC@name9}{subsubparagraph}
\@namedef{Map@TOC@name10}{subsubparagraph}
\@namedef{Map@TOC@name11}{subsubparagraph}
\@namedef{Map@TOC@name12}{subsubparagraph}
```

Continuing…

Typesetting a Map, Continued

`\Map@start` The macro `\Map@start` is used to initiate the use of a map. It takes no arguments. The map title is passed in via the macro `\Map@TITLE`.

It uses the `longtable` environment to perform the page breaking and marking of continued pages.

```
\newcommand\Map@start{%
  \advance\Map@counter1
  \setlength{\Map@length}{\textwidth}%
  \addtolength{\Map@length}{-\MapTitleFraction\textwidth}%
  \addtolength{\Map@length}{-\MapTextFraction\textwidth}%
  \ifx\Map@TITLE\empty\else
    \MapTOC{\Map@TITLE}%
  \fi
  \longtable
    {@{}p{\MapTitleFraction\textwidth}@{\hspace{\Map@length}}
        p{\MapTextFraction\textwidth}@{}}%
      \multicolumn{2}{@{}p{\textwidth}@{}}{%
        \MapFont{\MapTitleFont\rule{0pt}{3ex}%
          \Map@TITLE}}
    \endfirsthead
      \multicolumn{2}{@{}p{\textwidth}@{}}{%
        \MapFont{\MapTitleFont\rule{0pt}{3ex}%
          \Map@TITLE\MapContinuedFormat{\MapContinued}}}%
    \endhead
    \\
      &\MapRuleStart
       \rule{\MapTextFraction\textwidth}{\MapRuleWidth}\newline
       \mbox{}\hfill
       \raisebox{3pt}{\MapContinuingFormat{\MapContinuing}}
    \endfoot
      &\MapRuleStart
       \rule{\MapTextFraction\textwidth}{\MapRuleWidth}%
       \vspace{\MapParskip}
    \endlastfoot
    \xdef\@currentlabel{\Map@TITLE}%
    \label{Map@\the\Map@no}%
    \global\Map@open@true
}
```

Continuing…

Typesetting a Map, Continued

`\Map@end` The macro `\Map@end` is expanded when the end of the end of the `longtable` environment might be needed. The boolean `\ifMap@open@` determines whether such an environment is really open.

```
\newcommand\Map@end{%
  \ifMap@open@\vspace*{1.5ex}
    \global\Map@open@false
    \endlongtable
    \MapNewpage
  \fi
  \iftrue
   \ifnum\Map@blockcount>9
    \PackageWarning{limap}%
    {*** The current map contains too much blocks:
         \the\Map@blockcount}%
   \else\ifnum\Map@blockcount>7
    \PackageWarning{limap}%
    {--- The current map contains \the\Map@blockcount blocks.}%
   \fi\fi
  \fi
}
```

`\Map@UP` The macro `\Map@UP` contains the number of the parent map or the empty string.

```
\newcommand\Map@UP{}
```

`\Map@no` The counter `\Map@no` contains the sequence number for all maps. This value is used internally to reference single maps.

```
\newcount\Map@no
```

`\Map@counter` The counter `\Map@counter` contains the number of a map in the context of the containing map .

```
\newcount\Map@counter
\Map@counter=0
```

Continuing…

Typesetting a Map, Continued

`\Map@parts@` The macro `\Map@parts@` is used to store the parts of the top-level maps. This is the initialization of a feature otherwise used in the aux file.

```
289 \@namedef{Map@parts@}{}
```

Typesetting a Block

Blocks are the basic building unit of maps. Here the Block is defined in all it's beauty.

`Map@Block` This macro is used to typeset a block inside a Map. To avoid abuse outside of a map it is activated within a Map only.

```
\newenvironment{Map@Block}[1]{\par
  \vspace*{-\parskip}\vspace*{-1ex}%
  \\\null\par
  \vspace*{\MapParskip}%
  \raggedright\hspace{0pt}\MapFont{\MapBlockLabelFont{#1}}%
  \gdef\@currentlabel{#1}%
%  \global\advance\Map@blockcount1
  &\parskip=\MapParskip
  {\MapRuleStart
    \rule{\MapTextFraction\textwidth}{\MapRuleWidth}}\par
```

The final action is empty. Thus the block can be used as a simple macro as well.

```
}{%
}
```

`\Block` The macro `\Block` issues an error when used outside of a Map environment.

```
\newcommand\Block[1]{\PackageWarning{limap}{The sectioning command
      `Block' has been encountered outside the scope of a Map
      environment.}}
```

`\WideBlock` The macro `\WideBlock` takes one argument which is added to the current block where the whole width of the table is used.

```
\newcommand\WideBlock{\\\multicolumn2{@{}l@{}}}
```

Typesetting a Table of Contents

```
306 \newif\if@Map@toc@sep@
```

`\MapTableOfContents@open`

The macro `\MapTableOfContents@open` produces the table of contents for the current map. It produces a tabular containing the titles and pages of all maps directly contained in the current map. It utilizes a tabular environment and booktabs.

```
307 \def\MapTableOfContents@open{%
308   \centering
309   \begin{tabular}{p{.6\textwidth}r}\toprule
310     \MapTOCheadfont{\MapTOCname}&
311     \MapTOCheadfont{\MapTOCpage}\\\midrule
312     \ifcsdef{Map@parts@\the\Map@no}{
313       \edef\Map@tmp@{\csname Map@parts@\the\Map@no\endcsname:}%
314       \expandafter\Map@toc@loop\Map@tmp@
315       \\\bottomrule
316     }{}
317   \end{tabular}
318 }
```

`\MapTableOfContents@boxed`

The macro `MapTableOfContents@boxed` produces the table of contents for the current map. It produces a tabular containing the titles and pages of all maps directly contained in the current map. It utilizes a tabular environment and booktabs.

```
319 \def\MapTableOfContents@boxed{%
320   \centering
321   \begin{tabular}{|p{.6\textwidth}|r|}\hline
322     \MapTOCheadfont{\MapTOCname}&
323     \MapTOCheadfont{\MapTOCpage}\\\hline
324     \ifcsdef{Map@parts@\the\Map@no}{
325       \edef\Map@tmp@{\csname Map@parts@\the\Map@no\endcsname:}%
326       \expandafter\Map@toc@loop\Map@tmp@
327       \\\hline
328     }{}
329   \end{tabular}
330 }
```

Continuing…

Typesetting a Table of Contents, Continued

\MapTableOfContentsStyle

The macro MapTableOfContentsStyle determines the style of the TOCs. It can take the values open or boxed.

```
\newcommand\MapTableOfContentsStyle{open}
```

\MapTableOfContents

The macro MapTableOfContents produces the table of contents for the current map. It produces a tabular containing the titles and pages of all maps directly contained in the current map.

```
\newcommand\MapTableOfContents{\par
  \global\@Map@toc@sep@false
  \csname MapTableOfContents@\MapTableOfContentsStyle\endcsname
  \vspace*{-1.5\parskip}\par\ignorespaces
}
```

\Map@toc@loop

The macro \Map@toc@loop is a recursive solution to loop through all elements of a list of children. The argment is expected to be a colon separated list of numbers. The end is marked by two colons in row.

Each number is a reference to a map. The respective line in the toc table is produced.

```
\def\Map@toc@loop#1:{%
  \def\Map@tmp@{#1}%
  \ifx\Map@tmp@\empty
    \global\let\Map@next@=\relax
  \else
    \if@Map@toc@sep@
    \gdef\Map@next@{\\
    \ref{Map@#1}&\pageref{Map@#1}%
    \Map@toc@loop}%
    \else
    \gdef\Map@next@{%
    \ref{Map@#1}&\pageref{Map@#1}%
    \Map@toc@loop}%
    \global\@Map@toc@sep@true
    \fi
  \fi
  \Map@next@
}
```

Typesetting a Table

`\MapTabularFraction` The macro `\MapTabularFraction` contains a factor for the line width. This is used to determine the width of the table. The tabular is centered in the text column.

```
\newcommand\MapTabularFraction{.95}
```

`MapTabular` The environment `MapTabular` produces the `tabular` environment with the width of the text column.

```
\newenvironment{MapTabular}{%
  \begin{center}
    \begin{tabular*}{\MapTabularFraction\linewidth}%
}{%
    \end{tabular*}
  \end{center}\ignorespaces
}
```

Typesetting the Title Page

`\MakeTitle` The macro `\MakeTitle` can be used as a replacement for the `\maketitle` macro.

```
\newcommand\MakeTitle{\thispagestyle{empty}
  \rule{0pt}{.25\textheight}\par
  \mbox{}\hfill
  \begin{minipage}{\MapTextFraction\textwidth}
    \raggedright
    \rule{\textwidth}{2pt}\par
    \vspace*{2.5\MapParskip}%
    \sf{\huge \@title\par}%
    \vspace*{2.5\MapParskip}%
    \rule{\textwidth}{2pt}\par
    \vspace*{2.5\MapParskip}%
    \MapFont{\large \@author} \par
    \vspace*{2.5\MapParskip}%
    \MapFont{\footnotesize \@date}
    \vspace*{\MapParskip}%
  \end{minipage}%
  \vspace*{-22ex}%
  \par
}
```

`Abstract` This macro is used to typeset the abstract.

```
\newenvironment{Abstract}{\vfill
  \par
  \mbox{}\hfill
  \begin{minipage}{\MapTextFraction\textwidth}\parskip=1ex
    \rule{\textwidth}{1pt}\medskip\par

}{\par\rule{\textwidth}{1pt}
  \end{minipage}%
  \par
}
```

Continuing…

Typesetting the Title Page, Continued

Use maketitle and abstract

The new `\maketitle` macro is activated for the class.

```
391 ⟨*class⟩
392 \let\maketitle\MakeTitle
393 \let\abstract\Abstract
394 \let\endabstract\endAbstract
395 ⟨/class⟩
```

Final Actions

Local configuration

Load the configuration file at the end if it can be found.

```
\InputIfFileExists{limap.cfg}{}{}
```

Finally we have to activate the proper settings for the chosen language.

```
\csname LIMAP@SelectLanguage@\LIMAP@Language\endcsname
```

```
\ifLIMAP@strict
  \def\chapter{\PackageWarning{limap}{The sectioning command
      `chapter' is not available.}}
%  \def\section{\PackageWarning{limap}{The sectioning command
%      `section' is not available.}}
  \def\subsection{\PackageWarning{limap}{The sectioning command
      `subsection' is not available.}}
  \def\subsubsection{\PackageWarning{limap}{The sectioning command
      `subsubsection' is not available.}}
  \def\paragraph{\PackageWarning{limap}{The sectioning command
      `paragraph' is not available.}}
  \def\subparagraph{\PackageWarning{limap}{The sectioning command
      `subparagraph' is not available.}}
  \def\subsubparagraph{\PackageWarning{limap}{The sectioning command
      `subsubparagraph' is not available.}}
\fi
```

Finale

That's all for this time.

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.