

UNIVERSITY

Department

# Template of the Institute of Engineering Thermodynamics (M-21)

Vasco Alexander Wild, Alex Povel

Submitted in partial fulfillment of the requirements for the degree of Master of Science

**Advisor:** Advisor Name


**Committee:** Committee Member A, Committee Member B

**Defense Date:** November 2, 2025

---

City, Country

November 2, 2025

Name **main (CENSORED VERSION)**  
Compiled on **November 2, 2025 6:51pm Z**  
 **1125bed** (main)  
Engine LuaHBTeX, Version 1.22.0 (TeX Live 2025)  
L<sup>A</sup>T<sub>E</sub>X Version L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> (2025-06-01)  
Glossary glossaries-extra + bib2gls  
Bibliography biblatex + biber  
Generator latexmk  
Class 2025/09/09 v3.48 KOMA-Script

**Check for updates**

# Task

For: **Vasco Alexander Wild, Alex Povel** [666]

Topic: **Template of the Institute of Engineering Thermodynamics (M-21)** (Example Document)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

- First you are doing this,
- then another thing,
- lastly that.

You will be working on this a lot.

Prof. Dr.-Ing. Arne Speerforck

Rest of this page intentionally left blank.

## Declaration of Authorship

I, Vasco Alexander Wild, Alex Povel, hereby declare to be the sole, independent author of the Example Document submitted here. No other than the cited references have been used. Any content directly or indirectly obtained from external sources has been marked up as such. This thesis has neither been submitted to a second examination authority nor been published.

---

*Vasco Alexander Wild, Alex  
Povel*

---

*Place & Date*

Rest of this page intentionally left blank.

# Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing

elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

- We found this out,
- and also that!



# Contents

<b>Glossary</b>	<b>xix</b>
Symbols . . . . .	xix
Numbers . . . . .	xxi
Subscripts . . . . .	xxii
Acronyms . . . . .	xxii
<b>Glossary without page lists</b>	<b>xxv</b>
Symbols . . . . .	xxv
Numbers . . . . .	xxvii
Subscripts . . . . .	xxvii
Acronyms . . . . .	xxviii
<b>1. Preface</b>	<b>1</b>
<b>2. Usage</b>	<b>3</b>
2.1. Outside tools and special packages . . . . .	3
2.2. Using Docker . . . . .	5
2.2.1. Compilation . . . . .	6
2.2.2. More on Docker . . . . .	11
2.3. Editor . . . . .	13
2.3.1. Visual Studio Code . . . . .	14
2.3.2. Other Editors . . . . .	20
2.4. GitLab . . . . .	20

<b>3. Grundlagen</b>	<b>21</b>
3.1. Wissenschaftliches Arbeiten . . . . .	21
3.1.1. Ansprüche an die Wissenschaft . . . . .	22
3.1.2. Recherche . . . . .	23
3.1.3. Zitieren . . . . .	24
3.2. Typografische Richtlinien . . . . .	25
3.2.1. Schriftformatierung . . . . .	26
3.2.2. Befehle . . . . .	26
3.2.3. Konventionen . . . . .	26
3.2.4. Einheiten . . . . .	28
3.2.5. Abkürzungen . . . . .	30
 <b>4. Base Features</b>	 <b>31</b>
4.1. Fonts and Text . . . . .	32
4.1.1. Math . . . . .	34
4.1.2. Sans-Serif . . . . .	40
4.1.3. Mono-Spaced . . . . .	40
4.1.4. Symbols . . . . .	41
4.2. Language . . . . .	42
4.3. Sectioning . . . . .	43
4.3.1. Explanation . . . . .	43
4.4. References . . . . .	45
4.4.1. Bibliography . . . . .	45
4.5. Lists . . . . .	49
4.6. Censoring . . . . .	50
4.7. Glossaries . . . . .	51
4.7.1. bib2gls . . . . .	56
4.8. Landscape . . . . .	57

<b>5. Float Features</b>	<b>59</b>
5.1. TikZ and pgfplots . . . . .	68
5.1.1. Drawing over Bitmaps . . . . .	68
5.1.2. Trees . . . . .	68
5.1.3. Flow charts . . . . .	71
5.1.4. Plotting . . . . .	71
5.1.5. TikZ and Text . . . . .	77
5.1.6. Regular TikZ pictures . . . . .	77
5.1.7. InkScape . . . . .	83
5.2. Example Boxes . . . . .	83
<b>6. Code Syntax Highlighting</b>	<b>85</b>
6.1. Python . . . . .	86
6.2. MATLAB . . . . .	91
6.2.1. MATLAB/Simulink icons . . . . .	94
6.3. Modelica . . . . .	95
6.4. Lua . . . . .	96
<b>A. An appendix chapter</b>	<b>105</b>
A.1. More appendix . . . . .	105
A.2. Unprocessed data . . . . .	107
<b>B. Another appendix chapter</b>	<b>113</b>
<b>Index</b>	<b>119</b>

Rest of this page intentionally left blank.

# List of Figures

3.1.	Ansprüche an die Wissenschaft . . . . .	22
4.1.	Example for a censoring box . . . . .	51
5.1.	Example for a regular caption, spanning the whole width since it is so long. This can quickly look strange, especially if the figure itself is narrow . . .	62
5.2.	Example for a refined caption, spanning just the width of the float it is at- tached to, despite the caption being much longer than the float is wide . .	62
5.3.	An example for sub-figures . . . . .	63
5.3.	. . . . .	63
5.4.	A side caption, which may also span multiple lines like demonstrated in this rather long caption right here . . . . .	65
5.4.	. . . . .	65
5.6.	Drawing over a bitmap graphic using TikZ in a vertical sub-figure environ- ment . . . . .	69
5.7.	Example for a tree . . . . .	70
5.8.	Example for a Flowchart . . . . .	72
5.9.	Caloric parameters of air . . . . .	73
5.10.	Tufte-like plot . . . . .	73
5.11.	Wastegate feedback loop . . . . .	74
5.12.	Example automatic timeseries plot . . . . .	76
5.13.	Plotting from CSV data for a diffuser . . . . .	77
5.14a.	A vanilla MATLAB2TikZ example . . . . .	78
5.14b.	A MATLAB2TikZ plot recreated in L <sup>A</sup> T <sub>E</sub> X . . . . .	79

5.15a.	MATLAB2TikZ data replotted in $\LaTeX$ . . . . .	81
5.16.	Example for the <code>circuits.ee.IEC</code> TikZ library . . . . .	81
5.17.	Example for a three-dimensional TikZ drawing using the 3d library . . . . .	81
5.18a.	Example for a thermodynamic device drawing using TikZ. It relies heavily on the custom-made library of shapes . . . . .	82
5.18b.	Example TikZ shapes . . . . .	82
5.19.	Side-captions are still possible. So are labels . . . . .	84

# List of Tables

3.1.	Befehle zur Schriftformatierung im Textmodus . . . . .	27
3.2.	Befehle zur Schriftgrößenänderung . . . . .	27
3.3.	Konventionsvorschläge zur Text-Hervorhebung . . . . .	28
3.4.	Darstellung von Zahlen mit Einheiten . . . . .	29
3.5.	Darstellung von Zahlen und Einheiten mit <b>siunitx</b> -Paket . . . . .	29
3.6.	Abstände bei Abkürzungen . . . . .	30
4.1a.	Main/Roman Font Examples . . . . .	33
4.2.	Predefined math macros . . . . .	36
4.2a.	Sans-Serif Examples . . . . .	41
4.2b.	Monospaced Examples . . . . .	41
4.3.	Available languages and dynamic switching. Note how the commands are language-dependent; no manual adjustments necessary . . . . .	43
4.4.	Overview of Citation Commands and usage examples. . . . .	48
4.5.	Predefined glossaries . . . . .	55
5.1.	Example for multiple tables in one float . . . . .	64
5.2.	Compositions and properties of fuels, as used in Equation 5.1 . . . . .	66
5.3.	Dreadful version of Table 4.1a . . . . .	68
A.1.	A longtblr from the new <b>tabularray</b> package. . . . .	108

Rest of this page intentionally left blank.



# List of Examples

5.2.1. I am a useless box . . . . . 83

Rest of this page intentionally left blank.

# List of Code

6.2.	This is a page breaking code, with a caption, which is imported from a file!	86
6.1.	This is a caption. Listings cannot be overly long since floats do not page-break. Therefore is the longlisting environment. . . . .	87
6.4.	This is a page breaking matlab code, with a caption, which is imported from a file! . . . . .	91
6.3.	A class definition in MATLAB . . . . .	92

Rest of this page intentionally left blank.

# Glossary

*GlossaryLegend*

## Symbols

Roman	Description	Page List	Unit
$A$	Area	36, 54, 81	$\text{m}^2$
$C$	Heat capacity*	73	J/K
$c$	Velocity/Speed	34	m/s
$H$	Enthalpy*	34, 36	J
$H_i$	(specific) Heating Value*	40, 66	J/kg
$J_{\text{el.}}$	Electric current	81	A
$k$	Count/Quantity	34	–
$m$	Mass	36, 38, 81	kg
$M$	MACH number	77	–
$n$	Component	34	–

$p$	Pressure	34, 73, 77	Pa
$R$	(specific) Gas Constant*	34	J/(kg K)
$R_{el}$	Electrical resistance	81	$\Omega$
$Re$	REYNOLDS number	79	–
$r$	Radius	77	m
$T$	(absolute) Temperature	34, 36, 77	K
$t$	Time	36	s
$U_{el}$	Voltage	36, 81	V
$V$	Volume*	34, 36, 38, 81	m <sup>3</sup>
$x$	First Cartesian coordinate (length)	34, 36	m
$x$	Moisture Content	74	g <sub>w</sub> /kg <sub>da</sub> .
$y$	Second Cartesian coordinate (width)	36	m
$z$	Third Cartesian coordinate (height)		m
$\rho$	Density	34, 36, 38, 66, 77	kg/m <sup>3</sup>
<b>Greek</b>	<b>Description</b>	<b>Page List</b>	<b>Unit</b>
$\alpha$	Angle	34, 77	rad
$\gamma$	Mass fraction	40, 66, 77	–

$\zeta$	Drag Coefficient	79	–
$\kappa$	Ratio of specific heats	34, 73	–
$\varphi$	Relative Humidity		%RH
$\vartheta$	(relative) Temperature	34, 73	°C
<b>Other</b>	<b>Description</b>	<b>Page List</b>	<b>Unit</b>
$ \Box $	Absolute of $\Box$	36	$\Box$
$\bar{\Box}$	Arithmetic mean of $\Box$	36	$\Box$
$\dot{\Box}$	$\Box$ as a flow quantity	36, 81	$\Box/\text{s}$
$\Delta\Box$	Difference in $\Box$	34, 36	$\Box$
$d\Box$	infinitesimal change in $\Box$	34, 36	$\Box$
$\tilde{\Box}$	Logarithmic mean of $\Box$	36	$\Box$
$\partial\Box$	partial, infinitesimal change in $\Box$	34, 36	$\Box$
$\nabla\Box$	Vector of partial derivatives (Nabla operator) of $\Box$	36	–
$\mathbf{x}$	Vector	36	–

## Numbers

Symbol	Description	Physical Quantity	Page List
$e$	Euler's number	2.718 28...	34

$i$	Imaginary unit	$\sqrt{-1}$	34
$\pi$	Pi	3.141 59...	34, 54

Subscripts

<b>a.</b> air	<b>i</b> inferior
<b>d</b> dry	<b>th</b> thermal
<b>el.</b> electric	<b>w</b> Water

Acronyms

CD	Continuous Delivery	9
CI	Continuous Integration	9
COP	Coefficient of Performance	82
CTAN	Comprehensive T <sub>E</sub> X Archive Network	54
GUI	Graphical User Interface	
HFO	Heavy Fuel Oil	66



IDE	Integrated Development Environment	14, 15
IMO	International Maritime Organization	
ISO	International Organization for Standardization	41, 75
JSON	JavaScript Object Notation	15
MDO	Marine Diesel Oil	66
PDF	Portable Document Format	7, 10, 11, 19, 31, 59, 60
SSOT	Single Source of Truth	52, 59
SVG	Scalable Vectors Graphics	59, 60
URL	Uniform Resource Locator	40
WP	Wärmepumpe	55
WYSIWYM	What You See Is What You Mean	52

Rest of this page intentionally left blank.

# Glossary without page lists

*The following styles do not contain page lists of the entries' occurrences, leading to a cleaner, more concise look. Refer to the source code on how to achieve this (which options and styles to use).*

## Symbols

Roman	Description	Unit
$A$	Area	$\text{m}^2$
$C$	Heat capacity*	J/K
$c$	Velocity/Speed	m/s
$H$	Enthalpy*	J
$H_i$	(specific) Heating Value*	J/kg
$J_{\text{el.}}$	Electric current	A
$k$	Count/Quantity	–
$m$	Mass	kg
$M$	MACH number	–
$n$	Component	–
$p$	Pressure	Pa

$R$	(specific) Gas Constant*	J/(kg K)
$R_{\text{el}}$	Electrical resistance	$\Omega$
$Re$	REYNOLDS number	–
$r$	Radius	m
$T$	(absolute) Temperature	K
$t$	Time	s
$U_{\text{el}}$	Voltage	V
$V$	Volume*	$\text{m}^3$
$x$	First Cartesian coordinate (length)	m
$x$	Moisture Content	$\text{g}_w/\text{kg}_{\text{da}}$
$y$	Second Cartesian coordinate (width)	m
$z$	Third Cartesian coordinate (height)	m
$\rho$	Density	$\text{kg}/\text{m}^3$
<b>Greek</b>	<b>Description</b>	<b>Unit</b>
$\alpha$	Angle	rad
$\gamma$	Mass fraction	–
$\zeta$	Drag Coefficient	–
$\kappa$	Ratio of specific heats	–
$\varphi$	Relative Humidity	% RH
$\vartheta$	(relative) Temperature	$^{\circ}\text{C}$
<b>Other</b>	<b>Description</b>	<b>Unit</b>
$ \boxminus $	Absolute of $\boxminus$	$\boxminus$

$\bar{x}$	Arithmetic mean of $x$	$x$
$\dot{x}$	$x$ as a flow quantity	$x/s$
$\Delta x$	Difference in $x$	$x$
$dx$	infinitesimal change in $x$	$x$
$\tilde{x}$	Logarithmic mean of $x$	$x$
$\partial x$	partial, infinitesimal change in $x$	$x$
$\nabla x$	Vector of partial derivatives (Nabla operator) of $x$	–
$\mathbf{x}$	Vector	–

Numbers

Symbol	Description	Physical Quantity
$e$	Euler’s number	2.718 28...
$i$	Imaginary unit	$\sqrt{-1}$
$\pi$	Pi	3.141 59...

Subscripts

<b>a.</b> air	<b>d</b> dry
	<b>el.</b> electric

**i** inferior**w** Water**th** thermal

## Acronyms

Notation	Description
CD	Continuous Delivery
CI	Continuous Integration
COP	Coefficient of Performance
CTAN	Comprehensive T <sub>E</sub> X Archive Network
GUI	Graphical User Interface
HFO	Heavy Fuel Oil
IDE	Integrated Development Environment
IMO	International Maritime Organization
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
MDO	Marine Diesel Oil

<b>Notation</b>	<b>Description</b>
PDF	Portable Document Format
SSOT	Single Source of Truth
SVG	Scalable Vectors Graphics
URL	Uniform Resource Locator
WP	Wärmepumpe
WYSIWYM	What You See Is What You Mean

Rest of this page intentionally left blank.



# 1. Preface

This document is not a beginner guide. There is a wide choice of those out there already, both free and paid for. However, what is lacking is a collection of modern, or even at least current, best practices. If you scouted through package documentations and [StackExchange](#) long enough, you would eventually get an idea of what is current, idiomatic  $\text{\LaTeX}$  and what is not. This is how I learned  $\text{\LaTeX}$  too, so I cannot recommend any useful beginner books. You might want to check out [Overleaf](#) though, an online  $\text{\LaTeX}$  editor<sup>1</sup> with a large knowledge database.

This document is an attempt at collecting best practices — or at least, useful approaches — and pointing out the old ones they could, often should, and sometimes absolutely have to replace. See also [here](#) for a second opinion on the topic. More than most other languages, the  $\text{\LaTeX}$  code in circulation world-wide is quite aged. While that code does not necessarily get *worse*, it also does not exactly age like cheese and wine would. To that end, notice how the packages integral to this document are actively maintained and kept modern.

**Usage as a Cookbook** This document is supposed to be used in a *cookbook* style. That is, it is not meant to be read cover to cover (declaring it a cookbook is also a useful excuse for the mess I made).

The document contains various “recipes”, most of which exist in parallel and are independent of one another. The goal is to answer questions of the form *How can this and that be done?* A quick search in the (printed) output or the source code is supposed to deliver an-

---

<sup>1</sup>I do not recommend using online editors. You are putting your hard work onto some remote, foreign server, relying on their ongoing availability and forfeiting the chance of understanding  $\text{\LaTeX}$ , in case you have to continue locally. Theses containing confidential material should also not be hosted externally.

swers.

**Usage as a Template** Considerable effort went into the class file, aka the template. It pulls all settings together, producing an output that is very different from vanilla  $\text{\LaTeX}$ . Therefore, feel free to rip out all the cookbook content, keeping just the settings files to be used as a template.

**Source Code** This document is meant to be read side-by-side with its source code. That is why there is almost no source code in the printed output itself. If you are curious how a certain output is achieved, navigate to the source code itself. This approach was chosen since, plainly, code does not lie, while annotations and comments might. So while the printed output will always remain true to its actual source code, duplicating that source code so it can be read in the printed output directly is just another vector for errors to creep in.

**Source Repository** The source repository for this document is at

<https://collaborating.tuhh.de/m21/public/theses/latex-cookbook>.

Any references to the “source” or “repository” refer to that project.

## 2. Usage

This document “requires”<sup>1</sup> [Docker](#). On a high level, Docker allows to prepare specific software bundles, tailor-made for whatever application the bundle author desires. Other users can then use these software bundles for their own projects. The bundles are well-defined and can be distributed and run very easily. The more complex and demanding the required software, the higher the benefit of using such bundles. The driving design principle and primary use case for these bundles is *isolation*. Whatever you do with the bundles, it happens in isolation from your host system. This allows a user to have, for example, arbitrary Python versions at their disposal, whereas a local system installation can only ever offer a single version.

For a  $\text{\LaTeX}$  document, the one at hand here is pretty complex. This is owed to the many used packages and outside tooling. Outside tooling (programs other than  $\text{\LaTeX}$  called from within  $\text{\LaTeX}$ ) is quite prevalent in  $\text{\LaTeX}$ , since it itself is so limited.<sup>2</sup> Let us look at the outside tooling used for this document.

### 2.1. Outside tools and special packages

This section highlights the pain points of *not* using Docker. If you are already familiar and need no convincing, skip to [Section 2.2](#). For this document, outside tools are required for:

---

<sup>1</sup>It does not *technically* require Docker, but I hope to convince you that the non-Docker, manual way is the way of the dodo and a big no-no.

<sup>2</sup>For example, when writing Python, you would not call Perl or JavaScript from within it, because whatever they can do, Python can. The same analogy does *not* hold for  $\text{\LaTeX}$ : base  $\text{\LaTeX}$  can do surprisingly little, even though  $\text{\TeX}$  is technically Turing-complete.

1. **glossaries-extra**: requires the outside tool **bib2gls**, see Section 4.7.1, which in turn requires a Java Runtime Environment.
2. **biblatex**: requires the outside tool **biber**, see Section 4.4.1.
3. **svg**: requires the outside program **InkScape**. Examples for that package's main command, `\includesvg`, are Figures 5.1 to 5.4 and 5.19.
4. Using `contour` `gnuplot` commands with **pgfplots**, see ??, requires **gnuplot**.
5. Syntax highlighting for source code, see Chapter 6, is done through **minted**. It requires **Python** (built into virtually all Linux distributions but needs to be installed on Windows) with its **pygments** package installed. That package does the heavy lifting and **minted** inserts the result into your  $\text{\LaTeX}$  document.

If you have a proper  $\text{\LaTeX}$  distribution, **bib2gls** and **biber** will already be available as commands and ready to be invoked. The latter means that they are on your `$PATH`, *i.e.* the paths to the respective binaries are part of the `PATH` environment variable. *All other stuff, you have to install manually.*

You will have to install and keep everything else updated by hand. What if one of the dependencies of this document conflicts with something you have already installed on your system? What if that conflict is not resolvable? You would be unable to use this document. This is where Docker comes into play. You outsource all this setting-up to someone who already bundled all that stuff together. Docker calls these “bundles” *images*. There is a Docker image *tailor-made* for this document:

$\text{\LaTeX}$  Dockerfile of the ITT.

It is guaranteed to function correctly for this document, since the author maintains both in parallel. There will never be a mismatch. If maintenance ceases, it will cease for both components at the same time, hence it will still continue to work, just not get updated anymore.

## 2.2. Using Docker

Instead of all of the above, **only one installation is required**: Docker. You can get it from

<https://docs.docker.com/get-docker/> .

*You don't even need a  $\text{\LaTeX}$  distribution.* All you need is an editor to edit the files in, see [Section 2.3](#). With Docker installed, you can decide between two opportunities:

1. Using Docker itself to compile
2. Using [Visual Studio Code](#) to compile (recommended for beginners, see [Section 2.3](#))

In this section the first opportunity will be explained and in [Section 2.2.1](#) the different compilation methods will be explained. If you want to use [Visual Studio Code](#) you should just read [Section 2.2.1](#) to understand the different compilation methods, but don't worry about the terminal Code. The rest can be skipped and you proceed with [Section 2.3](#).

Once you want to compile, open a terminal, navigate to your directory and run

```
| docker run --rm --volume ${PWD}:/tex "ghcr.io/wyattau/omnilatex-docker:latest"
```

for PowerShell or

```
| docker run --rm --volume $(pwd):/tex "ghcr.io/wyattau/omnilatex-docker:latest"
```

for bash. That's it!

The command consists of:

- The `--rm` option, which removes the run container after you are done (containers are “instances” of images). This is generally desired, since containers are ephemeral and should be treated as such. Do not keep containers around, simply create new ones! If you need adjustments, adjust the image, not the container, then create new containers from that adjusted image.

- The `--volume` option makes the current directory (`pwd`) available *inside* the running container, at the `/tex` destination path. The  $\text{\LaTeX}$  process needs your files to work with. Without this option, the container would be “naked”, with no way of accessing your files.
- The final argument to `docker run` is the image to be run, in this case `ghcr.io/wyattau/omnilatex-docker:latest`, which will use the image specified by the Link.

Ideally, the command can be registered as the “compilation” command of your editor. That way, you just hit the compile button and will be using Docker in the background, with no changes to your workflow.

### 2.2.1. Compilation

You are probably used to running `pdflatex` or similar on your source files, as many times as needed. So where does that step happen in the above `docker` command?

The Docker approach uses the **latexmk** tool to ease all the painful labour of running chains of `pdflatex`, `biber` *etc.* manually. `latexmk` automates  $\text{\LaTeX}$  compilation by detecting all the required steps and then running them as often as required. It requires **Perl**. Linux users will already have it available, Windows users may grab [Strawberry Perl](#). As such, this document’s processing pipeline *as a whole* requires Perl, although it is technically not required for document compilation only.

Once Perl is installed (of course, the Docker image already contains it), the entire document can be compiled by **simply calling latexmk**. You do not even have to provide a `*.tex` file argument. By default, `latexmk` will simply compile all found `*.tex` files. **The core ingredient to this magic process is the `.latexmkrc` configuration file**. You can find it in the repository root directory. It is tailored to this document and does not need to be touched if the compilation process itself has not changed. It also contains some more insights to the entire process.

`latexmk` is great because it figures out most things by itself and enjoys wide-spread acceptance and adoption. If it does not figure out everything from the get-go, it is easily cus-

tomized, like for this document.

Having walked through all this manually, hopefully using the prepared Docker image instead makes more sense now. It is guaranteed to work for everyone, because the Docker container (that is, the virtual build environment) will be identical for all users. It is independent of local  $\text{\LaTeX}$  installations and all their quirks. As such, it simply and forever does away with the entire, huge class of

*But it works on my machine! (Everyone, at some point)*

Good riddance to that.

If all of this is embedded into a pipeline on GitLab, your documents are built whenever you `git push` to the remote server (or whenever you configure it to). For more information have a look at [Section 2.4](#). It does not get simpler; the downside is of course the lengthier setup, but all of that is explained in the [README](#). Also, the repository itself is a live demonstration where everything is set up already!

### Complete (slow) compilation

The most prevalent downside to `latexmk` are longer compilation times. The tool examines changes in auxiliary and source files. This is made possible by the way  $\text{\LaTeX}$  works with its kind of unique way of writing out auxiliary files (a system of multiple passes). Consider a simplified, manual example chain:

1. You have a `test.tex` file (and nothing else!), with `cref` or equivalent cross-referencing commands in it.
2. You compile the file *once*, e.g. using `lualatex`.
3. A `test.aux` file (and probably others, e.g. `.toc`) will have been generated, as well as the Portable Document Format (PDF) file itself.

The PDF has literal `??` markers where there should be references. This is because to resolve those references, and print e.g. the correct page number,  $\text{\LaTeX}$  needs to look into the `.aux` file. But that file was just generated and was not available yet for the first compilation run.

#### 4. Hence, you compile again.

The ?? markers are now replaced by their properly spelled-out reference. However, what if, hypothetically, one of these ?? substitutions now reads *very super long reference to page 37 in the document*. It is so long that subsequent content has shifted again. That content is now on different pages. This triggered another change in the auxiliary files.

#### 5. You compile *again*, the third time. Things have settled, no changes happened this time anymore. You are done.

latexmk works by detecting this convergence towards a steady-state (which may, through bugs, never be reached). However, it would compile *an additional* time after [Item 5](#), to make sure or because it cannot really be certain (just talking from experience here, not knowledge). So while you gain deterministic, automated builds, they will take much longer.

What if you do not care if there are ?? markers left, undefined entries, shifted pages and so forth, maybe because you are only drafting things up? You can simply compile yourself once, manually, *e.g.* using `lualatex`. This will speed things up by a long shot, but is considered advanced usage. It does, as opposed to latexmk, not “just work” anymore, and beginners might wonder what those ?? are. Now that you know, you may handle them.

**Faster (*incomplete*) compilation** The base docker commands were shown at the beginning of [Section 2.2](#). An alternative is to run

```
docker run --rm --volume $(pwd):/tex
→ https://collaborating.tuhh.de/m21/public/theses/latex_dockerfile -e
→ '$max_repeat=1'
```

This limits latexmk to only run commands once (`-e` is for additional Perl code), with the following effects:

- latexmk might complain that it did not succeed for lack of repeats (including a non-zero exit code, meaning failure when it was only partial failure).



- Results are compiled quickly.
- `latexmk` and its tailor-made config is still used under the hood, so you do not have to adjust anything.
- Other commands like `biber` for bibliography generation might still be run.

This is probably your best bet to get faster drafting cycles while retaining high likelihood of it working.

**Fastest (*incomplete*) compilation** The following method is the most advanced (but not hard at all!). To overwrite the Docker image’s default, so-called **ENTRYPOINT**, which is `latexmk`, run the command as:

```
docker run --rm --volume $(pwd):/tex --entrypoint="lualatex"
→ https://collaborating.tuhh.de/m21/public/theses/latex_dockerfile --shell-escape
→ $FILENAME
```

Notice that you now have to give a filename, like `cookbook.tex`, since `lualatex` expects that. The above command is exactly like running *only* `lualatex`, just in the context of that image (with your files made available at `/tex`). The effects of this command are:

- No `latexmk` involved at all:
  - it is therefore the fastest approach,
  - but you have to pass all required options yourself (like `--shell-escape`).
- No commands like `biber` run; you have to do that yourself (or use `latexmk`).

This is the approach I personally use, while outsourcing the full `latexmk`-based compilations to a server using Continuous Integration (CI)/Continuous Delivery (CD). It affords us the fastest iteration speeds while having a full version available “in the background”: the best of both worlds and arguably the fastest workflow.

**Fastest still slow** We cannot compile faster than a naked `lualatex` run (docker overhead is negligible). For the current document you see here, even such a single run takes about one minute. Remember that you need upwards of three runs from a “cold” (no auxiliary files, empty caches, ...) start. Single, article-size, `pdflatex`-based documents compile in only a few seconds. The reasons for the slowness are, among others:

- `lualatex` is simply considerably slower than `pdflatex` (the additional capabilities make up for that many times over). One reason is the time spent dealing with (system) fonts.
- This document loads a metric crapton of packages, some of which are very expensive to load, like `tikz`. This is in the spirit of providing a useful document with numerous examples of various functionalities and packages. You should definitely **comment out the packages you do not need**.
- Compiling the entire document takes long, so **make use of the `include` and `subinclude` commands and comment out unneeded parts**. The more fine-grained you do this, the more control you get. If you cut it down to short (only a handful of PDF pages) files, you can work on and compile only those. This method has the highest time saving potential.
- If you perform `pgfplots` calculations within  $\text{\LaTeX}$ , *c.f.* Figures 5.9 and 5.10 and ??, expect significant slowness. For simple plots, this is still much better than having to use an external program, export data, import to  $\text{\LaTeX}$ , repeat on data change (instead of adjusting the equation in  $\text{\LaTeX}$  directly) *etc.*
- $\text{\LaTeX}$  is not compiled in parallel.

This document is like an aircraft carrier: modern, feature-rich and high-impact, but very sluggish.

**Print version** There was a time of this  $\text{\LaTeX}$  document where the print version was compiled separately, with an own  $\text{\LaTeX}$  Workshop recipe or an own GitLab pipeline job.

Since we now use the **ocgx2**-package (as add-on for **hyperref**), which colors links in the PDF file, but doesn't print them, this is no longer necessary.

### 2.2.2. More on Docker

You do not need to know of the entire chain of how Docker images are created and run. Only consuming the final image has all the benefits with little effort. However, the process is not complex:

1. A **Dockerfile text document** is created, containing instructions on how the image should look like (like what stuff to install, what to copy where, ...).

As a baseline, these instructions often rely on a Debian distribution. As such, all the usual Debian/Linux tools can be accessed, like `bash`.

An (unrelated) [example Dockerfile](#) can look like:

```

1  # Get the latest Debian Slim with Python installed
  FROM python:slim

  # Update the Debian package repositories and install a Debian package.
5  # Agree to installation automatically (`-y`)!
  # This is required because Dockerfiles need to run without user interaction.
  RUN apt-get update && apt-get install -y ffmpeg

  # Copy a file from the building host into the image
10 COPY requirements.txt .

  # Run some shell command, as you would in a normal sh/bash environment.
  # This is a Python-specific command to install Python packages according to
  ↪ some
  # requirements.
15 RUN pip install -r requirements.txt

  # Copy more stuff!
  COPY music-converter/ music-converter/

```

```

20 # This will be the command the image executes if run.
# It runs this command as a process and terminates as soon as the process ends
# (successfully or otherwise).
# Docker is not like a virtual machine: it is intended to run *one* process,
  ↳ then
# die. If you need to run it again, just create a new container (instance of a
25 # Docker image). Treat containers as *cattle*, not as a *pet*. The
# container-recreation process is light-weight, fast and the way to go.
#
# Of course, this does not stop anyone from running one *long-running* process
# (as in infinity, `while True`-style). This is still a good use-case for
  ↳ Docker
30 # (as are most things!). An example for this is a webserver.
ENTRYPOINT [ "python", "-m", "music-converter", "/in", "--destination",
  ↳ "/out" ]

```

The Dockerfile this project uses for LaTeX stuff is here:

[https://collaborating.tuhh.de/m21/public/theses/latex\\_dockerfile](https://collaborating.tuhh.de/m21/public/theses/latex_dockerfile)

It is not as simple, so not as suited for an example. Its length gives you an idea of the setup required to compile this  $\text{\LaTeX}$  document. All of that complexity is of no concern to you when using Docker! Of course, such an image also works for much simpler documents.

If you require custom additions, you can always inherit from existing base images:

```

1 FROM https://collaborating.tuhh.de/m21/public/theses/latex\_dockerfile

# ... Your stuff goes here ...

```

2. The **image** is then built according to the Dockerfile instructions, resulting in a large-ish file that contains an executable environment. For example, if we install a comprehensive TeXLive distribution, the image can be more than 2 GB in size. Note that you will never interact with that “file” directly. Docker manages it for you, and all interaction occurs through the `docker` command.

The Docker image can be distributed. If you just instruct to run an image called *e.g.*

`alexpovel/latex`, without specifying a full URL to somewhere, Docker will look on its Hub for an image of that name (and find it [here](#)). Anyone can pull (public) images from there, and everyone will be on the same page (alternatively, you could build the image from the Dockerfile, but this feature is deprecated/outsources to the [LaTeX Dockerfile Repo](#)). Because the Dockerfile this project uses can not be found on Dockerhub the full path/link to it has to be given.

For example, as stated, the  $\text{\LaTeX}$  environment for this project requires a whole bunch of setting-up. This can take more than an afternoon to read up upon, understand, implement and getting to run. In some cases, it will be impossible if some required part of a project conflicts with a pre-existing condition on your machine. For example, suppose project *A* requires Perl in version 6.9.0, but project *B* requires version 4.2.0. This is what Docker is all about: isolation. Whatever is present on your system does not matter, only the Docker image/container contents are relevant.

Further, if you for example specify `FROM python:3.8.6` as your base image, aka provided a so-called tag of 3.8.6, it will be that tag in ten years' time still. As such, you nailed the version your process takes place in and requires. Once set up, this will run on virtually any computer running Docker, be it your laptop now or whatever your machine is in ten years. This is especially important for the reproducibility of research.

3. Once the image is created, it can be run, **creating a container**. We can then enter the container and use it like a pretty normal (usually Linux) machine, for example to compile our  $\text{\LaTeX}$  files. Other, single commands can also be executed.

The proper way is to run one container *per process*. If that process (e.g. `latexmk`) finishes, the container exits. A new process then requires a new container.

## 2.3. Editor

You are free to do whatever you want. However, a garbage editor can substantially hamstring your work. For example, please do not use Notepad++. It is a fantastic little pro-

gram but unsuitable for any serious, longer work.

The author uses and dearly recommends [Visual Studio Code](#), using its [L<sup>A</sup>T<sub>E</sub>X Workshop](#) extension, which provides syntax highlighting, shortcuts and many other useful things. Visual Studio Code is among the most state-of-the-art editors currently available. Being usable for L<sup>A</sup>T<sub>E</sub>X is just a nice side-effect we can take advantage of. It is open-source and therefore also has a privacy-respecting alternative fork, [VSCodium](#). For a more conventional, complete Integrated Development Environment (IDE), try [TeXStudio](#). Like VSCode, it is also [open source](#). TeXStudio will cater to 90 % of your L<sup>A</sup>T<sub>E</sub>X needs, but nothing else (Markdown, ...).

### 2.3.1. Visual Studio Code

The repository of this document comes with a [.devcontainer](#) directory. In it is all the configuration necessary to run your development environment *inside* the Docker container entirely. To make the magic work, install the [Remote – Containers](#) extension. Visual Studio Code automatically prompts you to install it when you first open this repository, as configured in `.vscode/extensions.json`. Follow the instructions there on how to download all prerequisites, like of course Docker. Visual Studio Code will detect the remote container configuration of this repository automatically. If not, run the steps manually:

1. Open the command palette (`Ctrl` + `⇧` + `P` — the command palette is one of the defining, core features of Visual Studio Code, it's great!).
2. Run `Remote-Containers: Reopen in container`.

Your development environment is now *inside* the container. Hence, if you ran for example `latexmk` in the console, it executes the container version. This is exactly what we want and previously did using the `docker run` command.

## Extensions

There is a small problem. On its own, Visual Studio Code has no concept of  $\text{\LaTeX}$ , since out of the box, it is much closer to being an editor than an IDE. You increase the program's capabilities by choosing and installing extensions. Using the *LaTeX Workshop* extension for Visual Studio Code is recommended. For the container environment, it is installed automatically from the settings found in `devcontainer.json`. The *LaTeX Workshop* extension turns Visual Studio Code into a  $\text{\LaTeX}$  IDE. It works using *recipes*, which in turn use *tools*. The extension comes with a pre-defined `latexmk` tool:

1. Open the command palette.
2. Run `LaTeX Workshop: Build with recipe` `Complete compilation`.

This just runs `latexmk` within the container. Or you can click `TEX` on the left side of VSC and under `Build LaTeX project` select `Complete compilation`. As usual using Docker, your host machine does not need a  $\text{\LaTeX}$  installation for this to work. If you want, you can stop here — running `latexmk` is all that is ever needed. Hit `Ctrl + Alt + B` to build the project again, using the last used recipe.

## Settings

You can configure *everything* in Visual Studio Code, including its extensions, in the settings. Open them by either:

- The command palette at `Preferences: Open Settings (JSON)`, or
- navigating to `File` `Preferences` `Settings`, then opening as JavaScript Object Notation (JSON).

For example, as discussed at the end of [Section 2.2.1](#), you might want to have a recipe that only runs `lualatex`, once. This is **already taken care of** for this repository:

```
1 {
  "latex-workshop.latex.recipes": [
```

```

5      {
        "name": "Complete compilation",
        "tools": [
            "latexmk"
        ]
    },
    {
10      "name": "Faster Compilation",
        "tools": [
            "latexmk_once"
        ]
    },
15    {
        "name": "Fastest Compilation",
        "tools": [
            "lualatex"
        ]
    },
20 ],
    "latex-workshop.latex.tools": [
        {
            "name": "latexmk",
            "command": "latexmk",
            "args": [
                // latexmk doesn't need a file argument, but provide it to avoid
                ↪ running
                // on *all* found tex files
                "%DOC%"
            ],
30      },
        "env": {}
    },
    {
        "name": "latexmk_once",
        "command": "latexmk",
        "args": [
            // Don't know why, but the normal command/flag -e '$max_repeat=1'
35

```



```

// doesn't work because of the space. If it is seperated in two
// lines it works.
"--e",
"$max_repeat=1",
"%DOC%"
],
"env": {}
},
{
  "name": "lualatex",
  "command": "lualatex",
  "args": [
    "--shell-escape",
    "--synctex=1",
    "--file-line-error",
    "--halt-on-error",
    "--output-directory=build",
    "%DOC%"
  ],
  "env": {}
},
],
// Configure LaTeX Workshop to look for output in the build directory
"latex-workshop.latex.outDir": "./build",
// Make the first recipe the default recipe
"latex-workshop.latex.recipe.default": "first",
// Enable chktex for linting
"latex-workshop.linting.chktex.enabled": true,
// Run the linter on save
// Disable automatic build on save
"latex-workshop.latex.autoBuild.run": "never",
// Disable automatic retry of LatexWorkshop when an error occurred:
"latex-workshop.latex.autoBuild.cleanAndRetry.enabled": false,
// Set the latexindent command as the default formatter
"latex-workshop.formatting.latex": "latexindent",
// Packages which are always used by LaTeX Workshop for Intellisense, see:

```

```

// https://github.com/James-Yu/LaTeX-Workshop/wiki/Intellisense#latex-workshop
↪ intellisensepackageextra
75 // This is just a fallback. Because with this question asked:
↪ https://github.com/James-Yu/LaTeX-Workshop/issues/3965
// all packages from the class file shall be used. Just in case, the most
↪ important packages are listed here.
"latex-workshop.intellisense.package.extra": [
    "siunitx",
    "hyperref",
80 ],
// Be sure to make Latex-Workshop the standard formatter
"editor.defaultFormatter": "James-Yu.latex-workshop",
// Automatically format on save
"editor.formatOnSave": true,
85 // Make the .cls and .sty files to latex files, so they're displayed and
↪ compiled correctly
"files.associations": {
    "*.trsl": "latex",
    "*.cls": "latex",
    "*.sty": "latex"
90 },
// Set the encoding to utf8, to "gurarantee" that the files are opened and
// saved with the correct encoding
"files.encoding": "utf8",
// When opened in a container, automatically run the command
↪ "latex-workshop.compilerlog" to open the
95 // LaTeX compiler log. So that you are always confronted with the compiler log.
↪ See:
// https://stackoverflow.com/a/55191027 and
// https://marketplace.visualstudio.com/items?itemName=gabrielgrinberg.auto-run
↪ n-command
"auto-run-command.rules": [
    {
100     "condition": "isRunningInContainer",
    "command": "latex-workshop.compilerlog",
    "message": "Open LaTeX Compiler Log"
    }
  ]

```

105

```

    }
  ],
  // enable German dictionary
  "cSpell.language": "en,de-de",
}

```


These are *Workspace Settings*, which [take precedence over \(global\) user settings](#). As such, they also override all the recipes *LaTeX Workshop* usually comes with. You will have to define more recipes yourself.

With the new recipes, you can now run all three in [Section 2.2.1](#) described compilation methods:

- LaTeX Workshop: Build with recipe Complete compilation, or
- LaTeX Workshop: Build with recipe Faster compilation, or
- LaTeX Workshop: Build with recipe Fastest compilation, or

from the command palette. Or you can click TEX on the left side of VSC and under Build LaTeX project select your desired recipe. Notice the freedom and flexibility you have for defining many more recipes. In fact, alongside `latexmk` and classic `make`, you now have more potential flexibility and automation than one could possibly need. Using the above *Remote – Containers* approach, the entire development chain happens inside the container, with your working directory mounted into it. This enables:

- the *LaTeX Workshop* extension to work fully:
  - SyncTeX works:
    - \* Ctrl + Click into the PDF preview goes to the source code
    - \* LaTeX Workshop: SyncTeX from cursor (command palette) goes the other way.
  - Highlighting infos, warnings and errors (*Problems* pane) works
  - [IntelliSense](#) works, providing autocompletion for:
    - \* citations (autocite *etc.*) – check out the *Citation Browser* in the command palette!

- \* references (`\cref` *etc.*)
- \* and more, like commands
- Auto-Formatting (`( + Alt + F)`) works (*LaTeX Workshop* uses `\latexindent`)
- Your files are persisted normally, despite the container being ephemeral.
- Tools like `make` work from within the Visual Studio Code terminal. For example, run `make clean` to clear everything not tracked by git. This is very convenient for a cold start, because  $\text{\LaTeX}$  sometimes gets stuck if auxiliary files are corrupted.

### More Settings

For more control, you might want to keep the extension from compiling on every document save:

```
1 | "latex-workshop.latex.autoBuild.run": "never"
```

See [the official documentation](#) for more.

### 2.3.2. Other Editors

You will have to incorporate the `docker` commands shown in [Section 2.2](#) manually. Most editors do not support container-native developing, so you will go the old-school route.

## 2.4. GitLab

It is highly recommended to use this cookbook in combination with GitLab. Therefore, you just fork the GitLab Repo of this cookbook. Then you just have to install git and pull this Repo afterwards. If you do so, you have many nice features enabled from which you can profit. For example everytime you push a new version to GitLab, the CI pipeline compiles the whole document for you. And of course, like this you have a version control and backup of your document. So, please consider using GitLab.

## 3. Grundlagen

In diesem Chapter geht es um die Grundlagen des wissenschaftlichen Arbeitens und die Typographischen Grundlagen für diese.

Dieser Teil stammt aus der “alten” Institutsvorlage und ist daher nicht unbedingt top aktuell was Aussagen über das Template betrifft. Hierfür sind die anderen Kapitel und Abschnitte am Besten geeignet. Allerdings wird hier noch einmal auf das wissenschaftliche Arbeiten und generelles was Typographie betrifft eingegangen, weswegen dieser Teil hier aufgenommen wurde.

### 3.1. Wissenschaftliches Arbeiten

Ziel des wissenschaftlichen Arbeitens ist der Gewinn neuer Erkenntnisse. Insbesondere müssen Probleme strukturiert, gegliedert und methodisch und systematisch gelöst werden [Voss 2011]. Problemlösungsprozesse sind Teil des Studiums und müssen für studentische Arbeiten eigenständig angewandt werden. Daneben geht es beim wissenschaftlichen Arbeiten auch um eine entsprechende Form der schriftlichen Ausarbeitung und der Präsentation.

In dieser Vorlage werden nur einige Aspekte aus dem wissenschaftlichen Arbeiten hervorgehoben: die Ansprüche an die Wissenschaft, die Recherche und das Zitieren. Die Form der schriftlichen Darstellung wird mit der Vorlage bereits geliefert und ausführlich besprochen, so dass hierauf nicht weiter eingegangen werden muss. Für allgemeinere Betrachtungen zum wissenschaftlichen Arbeiten sei auf die Fachliteratur verwiesen: Literaturempfehlung deutsch:

- [VOSS 2011]
- [SESINK 2007]
- [KARMASIN and RIBING 2010]

### 3.1.1. Ansprüche an die Wissenschaft

Bei den studentischen Arbeiten handelt es sich um wissenschaftliche Arbeiten, für die gewisse, allgemeine Kriterien erfüllt werden müssen. Nach [Voss 2011] lassen sich diese Ansprüche in folgende Adjektive gliedern: *objektiv*, *präzise*, *zuverlässig*, *vollständig*, *ehrlich* und *ethisch*, siehe auch Figure 3.1.

Diese Kriterien werden nun kurz umrissen, wobei hierfür die Erläuterungen aus [Voss 2011] übernommen werden. *Objektivität* bedeutet, dass eine möglichst neutrale und analysierende Position zum bearbeiteten Thema eingenommen wird. Die gewonnenen Resultate sollten *präzise*, d. h. eindeutig und verständlich für den Leser sein. Ergebnisse sind *zuverlässig*, wenn bei wiederholten Untersuchungen (bspw. Messungen) jeweils die gleichen Ergebnisse auftreten. Dieses Kriterium ist nicht immer erfüllbar, da eine Situation geschaffen werden muss, in der alle Einflussfaktoren konstant bleiben. Die *Vollständigkeit* bedeutet, dass alle relevanten Aspekte der Aufgabenstellung untersucht werden und die Informationsgrundlagen (z. B. durch Literaturrecherchen) möglichst ausgeschöpft werden. Auch dieses Kriterium ist für studentische Arbeiten nicht immer

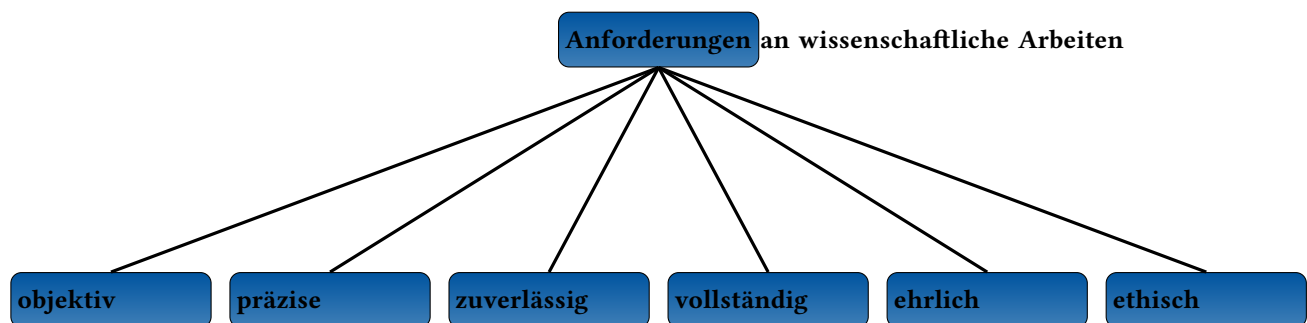


Figure 3.1: Ansprüche an die Wissenschaft.

komplett erfüllbar, da hier auch der zeitliche Rahmen eine Rolle spielt. Das Kriterium *ehrlich* bedeutet, dass der Autor alle Quellen, aus denen Erkenntnisse, Argumente und Anregungen gewonnen hat, offen legt. Für dieses Kriterium ist also das korrekte Zitieren wichtig, siehe [Section 3.1.3](#). Zu guter Letzt sollte die Wissenschaft *ethisch* sein, d. h. sich an Standards der Menschlichkeit, Würde und Erhaltung der Umwelt orientieren.

### 3.1.2. Recherche

In der Regel beginnt eine studentische Arbeit mit der Recherche und Quellensuche. Hierfür sollen einige Tipps genannt werden, um vorhandene Informationen leichter zu finden. Es empfiehlt sich für die Recherche folgende Quellen/Suchmöglichkeiten zu berücksichtigen:

- **Betreuer der Arbeit oder Fachexperte:**

Zunächst sollte der Betreuer nach entsprechender Literatur oder Informationen zum Thema gefragt werden. Meistens ist bereits einiges zum Thema vorhanden oder zumindest das Wissen, welche Literatur in Frage kommt. Neben dem Betreuer kommen etwaige Fachexperten aus Universität oder Industrie in Frage, um erste Informationen zu sammeln.

- **Studentische Arbeiten oder Veröffentlichungen des Instituts:**

Am Institut gibt es eine Vielzahl von studentischen Arbeiten oder Veröffentlichungen. Häufig sind Arbeiten vorhanden, die eine ähnliche Thematik oder Methodik verwendet haben. Hiermit ist ein guter Startpunkt für die eigene Arbeit gegeben.

- **Tagungsbände (Proceedings) von Fachkonferenzen:**

Bei sehr neuen Technologien ist Standardliteratur meist rar. Eine gute Quelle können Tagungsbände von Konferenzen sein. Hier werden von den Experten die neuesten Ergebnisse vorgestellt. Fragt euren Betreuer nach den Namen entsprechender Konferenzen.

- **Bibliotheks- oder Verbundkataloge:**

Neben dem Katalog der eigenen Universitätsbibliothek [TUB] sind vor allem auch Verbundkataloge von Interesse. Diese listen alle verfügbaren Titel eines Bibliothekenverbundes auf. Im norddeutschen Raum ist der Katalog des Gemeinsamen Bibliotheksverbundes [GBV] die zentrale Anlaufstelle und kann über eine Fernleihe genutzt werden.

- **eBooks der Verlage:**

Über das TUHH-interne Intranet gibt es Vollzugriff auf die eBooks unterschiedlicher Verlage, wie z. B. [SpringerLink](#) oder [Oldenbourg-Link](#). Eine Vielzahl weiterer eBook-Angebote stehen zur Verfügung. Weitere Informationen sind der [Homepage der Universitätsbibliothek](#) zu entnehmen.

- **Internetsuchmaschinen:**

Zu guter Letzt spielt natürlich das Internet in der Recherche eine immer wichtigere Rolle. Für wissenschaftliche Zwecke kommen beispielsweise Suchmaschinen wie [Google Scholar](#), [Google Books](#) oder [Bielefeld Academic Search Engine \(BASE\)](#) in Frage. Informationen direkt aus dem Internet zu übernehmen sollte, wenn möglich, vermieden werden. Zum einen ist die Zugänglichkeit der Informationen nicht langfristig gewährleistet und zum anderen ist die Zitierfähigkeit häufig nicht gegeben (Beispiel: Wikipedia).

Diese Auflistung ist natürlich nur eine Auswahl an Möglichkeiten. Wichtig ist es, sich ein möglichst umfassende Sammlung relevanter Quellen zuzulegen, um bekannte Erkenntnisse zum Thema verwerten zu können. Wichtig ist aber eher die Qualität der Quellen, als die Quantität.

### 3.1.3. Zitieren

Auch wenn wissenschaftliche Arbeiten die Eigenleistung des Autors wiedergeben sollen, ist es unabdingbar bereits vorhandenes Wissen zu verwenden und darzustellen. Es ist jedoch äußerst wichtig, dass man Eigenleistung und fremdes Wissen deutlich unterscheidbar macht. Aus diesem Grund muss fremdes Wissen **immer** mit Quellen belegt



werden und zwar so, dass auch eine eindeutige Zuordnung zur Quelle möglich ist. Hierdurch wird eine Nachprüfbarkeit der Aussagen der Arbeit ermöglicht [Voss 2011]. Es werden zwei Arten von Zitaten unterschieden: direkte und indirekte Zitate. “Direkte Zitate sind wortwörtliche Übernahmen aus fremden Texten” [Voss 2011]. Sie müssen grundsätzlich in Anführungszeichen gesetzt werden. Sie werden verwendet, wenn der getreue Wortlaut wichtig ist und hervorgehoben werden soll [Voss 2011]. Bei indirekten Zitaten handelt es sich um die sinngemäße Wiedergabe von Aussagen, bei denen keine Anführungszeichen verwendet werden [Voss 2011]. Die Form der Quellenangabe wird durch diese Vorlage vorgegeben (durch den `\cite{bibid}`-Befehl), so dass sich der Autor keine weiteren Gedanken darüber machen muss.

## 3.2. Typografische Richtlinien

In diesem Section werden typografische Richtlinien festgelegt bzw. vorgeschlagen, die eine übersichtlichere Gestaltung und eindeutige Formatierung des Dokuments bewirken. Allgemein sollte darauf geachtet werden, typografische Richtlinien konsistent über das gesamte Dokument anzuwenden. Dazu empfiehlt es sich eigene Makros in der Datei *main.tex* zu definieren und diese zu verwenden. Eine Änderung der Formatierung im Makro wird dann automatisch im gesamten Dokument übernommen.

Im [Section 3.2.1](#) wird erläutert, auf welche Weise Textelemente bzgl. ihrer Schrift formatiert werden. Hierdurch können beispielsweise besondere Wörter wie Eigennamen hervorgehoben werden können. Ein Vorschlag für eine Konvention zur Formatierung von Textelementen wird gegeben.

In den folgenden Abschnitten [Section 3.2.4](#) bis [Section 3.2.5](#) wird die korrekte Formatierung von Einheiten und Abkürzungen erläutert. Hierbei handelt es sich um anerkannte typografische Regeln, die auf jeden Fall eingehalten werden sollten.

### 3.2.1. Schriftformatierung

Über die Schriftformatierung können Textelemente besonders ausgezeichnet werden, beispielsweise durch eine *kursive* Darstellung. Im Folgenden werden grundlegende Informationen daraus präsentiert. Für weitere Details sei auf L<sup>A</sup>T<sub>E</sub>X<sub>3</sub> PROJECT TEAM (LPT<sub>05</sub>) verwiesen.

### 3.2.2. Befehle

Für die Schriftformatierungen stehen die in Table 3.1 gezeigte Befehle zur Verfügung. Aus vorherigen  $\text{\textit{E}T}_{\text{\textit{X}}}$ -Versionen stehen veraltete Befehle (z. B. `\bf`) zur Verfügung, die jedoch nicht mehr genutzt werden sollten. Es stehen jeweils zwei Varianten bereit. Die `\textxx{...}`-Befehle formatieren den Text innerhalb der Klammer, die weitere Variante ist ein Schalter, der den gesamten nachfolgenden Text (innerhalb der jeweiligen Umgebung) formatiert.

Die Schriftgröße kann über die Befehle in Table 3.2 verändert werden. Die Befehle sind keine Absolutangaben sondern als relative Änderung im Vergleich zur Standardschriftgröße zu verstehen. Für die hier verwendete Standardgröße von 12 pt realisiert der Befehl `\huge` bereits die maximal mögliche Größe, so dass im Vergleich dazu `\Huge` keine Veränderung mehr bewirkt.

### 3.2.3. Konventionen

Änderungen der Schriftformatierung sollten sehr bedacht und in eher geringem Maße angewandt werden. Wichtig ist auch die Konsistenz bei der Anwendung. In diesem Abschnitt soll ein *Vorschlag* für Konventionen präsentiert werden, der sich an häufig verwendeten Formatierungen richtet, jedoch keine vorgeschriebene Regel darstellt, siehe Table 3.3. Der Autor (ggf. auch der Betreuer) ist frei in der Wahl, ob diese Konventionen verwendet und ggf. auf eigene Bedürfnisse angepasst werden sollen!

Die Grenzen zum Anwendungsbereich der einzelnen Formatierungsvorschläge sind schwammig und obliegen dem Autor. Ggf. ist es auch sinnvoll eigene Makros zu definieren.

**Table 3.1:** Befehle zur Schriftformatierung im Textmodus [LPT05] .

Formatierung	Befehl	Beispiel
Roman (Serifenschrift)	<code>\textrm{...}</code> od. <code>\rmfamily</code>	Beispiel abc ABC 123
Bold Face (fett)	<code>\textbf{...}</code> od. <code>\bfseries</code>	<b>Beispiel abc ABC 123</b>
Italics (kursiv)	<code>\textit{...}</code> od. <code>\itshape</code>	<i>Beispiel abc ABC 123</i>
Slanted (schräggestellt)	<code>\textsl{...}</code> od. <code>\slshape</code>	<i>Beispiel abc ABC 123</i>
Sans Serif (serifenlose Schrift)	<code>\textsf{...}</code> od. <code>\sffamily</code>	Beispiel abc ABC 123
Small Caps (Kapitälchen)	<code>\textsc{...}</code> od. <code>\scshape</code>	BEISPIEL abc ABC 123
Typewriter (Schreibmaschine)	<code>\texttt{...}</code> od. <code>\ttfamily</code>	Beispiel abc ABC 123

**Table 3.2:** Befehle zur Schriftgrößenänderung [LPT05] .

Größe (ca.)	Befehl	Beispiel
50%	<code>\tiny</code>	Beispiel
70%	<code>\scriptsize</code>	Beispiel
80%	<code>\footnotesize</code>	Beispiel
90%	<code>\small</code>	Beispiel
100%	<code>\normalsize</code>	Beispiel
120%	<code>\large</code>	Beispiel
140%	<code>\Large</code>	Beispiel
170%	<code>\LARGE</code>	Beispiel
210%	<code>\huge</code>	Beispiel
250%	<code>\Huge</code>	Beispiel

**Table 3.3:** Konventionsvorschläge zur Text-Hervorhebung .

Objekt	Formatierung	Beispiel
Personen-, Firmennamen etc.	Kapitälchen	ISAAC NEWTON
Produkt-, Modellbezeichnungen	schräggestellt	<i>Airbus A380</i>
Allg. Hervorhebung, Fremdwörter etc.	kursiv	<i>a priori</i>
Auffällige Hervorhebung (eher nicht verwenden!)	fett	<b>sehr wichtig</b>
Besondere Bezeichnung, umgangssprachlich etc.	Anführungszeichen	“trivial”

Beispielweise ein Makro `\person{}`, das die Schriftformatierung von Personen und Eigennamen umsetzt. Vorteil hierbei ist, dass eine Änderung der Schriftformatierung einmalig global erfolgt. Makros können in der Präambel der *main.tex* definiert werden.

### 3.2.4. Einheiten

In dieser Vorlage sollte am Besten alles so umgesetzt werden wie in [Section 4.1.1](#) beschrieben. Nichtsdestotrotz gelten die hier beschriebenen Konventionen.

Bei der richtigen Darstellung von Zahlenwerten mit Einheiten gibt es einige Regeln zu beachten. Zum einen wird die Einheit immer mit aufrechter Schrift dargestellt [[PTBo7](#)]. Zum anderen ist ein entsprechender Abstand zwischen Zahlenwert und Einheit notwendig, der jedoch nicht so groß wie ein normales Leerzeichen sein sollte. Weiterhin muss verhindert werden, dass Zeilenumbrüche zwischen Zahlenwert und Einheit stattfinden. Beispiele für richtige und falsche Darstellungsweisen sind in [Table 3.4](#) gegeben.

Statt der manuellen Formatierung der Abstände und Einheitenschrift wird empfohlen das Paket `siunitx` zu verwenden. Dieses bietet weitere Funktionalitäten wie die Gruppierung von 3er-Zahlenblöcken, Konvertierung von Dezimaltrennzeichen etc. an. Die Verwendung erfolgt über die Befehle `\SI{...}` bzw. `\num{...}`. Einige Beispiele sind in [Table 3.5](#) gezeigt. Für weitere Informationen bitte das [Benutzerhandbuch](#) verwenden.

**Table 3.4:** Darstellung von Zahlen mit Einheiten .

richtig		falsch		
Code	Darstellung	Code	Darstellung	Hinweis
<code>1\,m</code>	1 m	<code>1m</code>	1m	Kein Abstand
<code>\SI{1}{\meter}</code>	1 m	<code>1 m</code>	1 m	Abstand zu groß
<code>\(1\,\mathrm{m}\)</code>	1 m	<code>\(1 m\)</code>	1m	Einheit kursiv

**Table 3.5:** Darstellung von Zahlen und Einheiten mit **siunitx**-Paket .

siunitx-Befehl	Darstellung	Hinweis
<code>\SI{1}{\meter}</code>	1 m	Abstand/Schrift automatisch (Text-Modus)
<code>\SI{1}{\meter \per \square \second}</code>	1 m/s <sup>2</sup>	Verwenden von SI-Einheiten Namen
<code>\SI{30}{\relhumidity}</code>	3 % RH	Eigene Einheit für rel. Feuchte, abhängig von Sprache
<code>\SI{1.1(5)}{\meter}; \SI{1,2+-5}{\meter}</code>	(1.1 ± 0.5) m; (1.2 ± 5.0) m	Konvertierung Ungenauigkeiten richtige Darstellung mit Einheiten
<code>\(\SI{1}{m}\)</code>	1 m	Abstand/Schrift automatisch (Mathe-Modus)
<code>\SI{100}{\frac{m}{s}}</code>	100 $\frac{\text{m}}{\text{s}}$	Bruchdarstellung der Einheit möglich
<code>\SI{100}{\dfrac{m}{s}}</code>	100 $\frac{\text{m}}{\text{s}}$	Bruchdarstellung mit normaler Schriftgröße
<code>\num{10000}</code>	10 000	Gruppierung von 3er-Zahlenblocks
<code>\num{1000}</code>	1000	Keine Gruppierung bei 4er-Zahlenblock
<code>\num{1.1}; \num{1,2}</code>	1.1; 1.2	Konvertierung Dezimaltrennzeichen
<code>\num{1.1(5)}; \num{1,2+-5}</code>	1.1 ± 0.5; 1.2 ± 5.0	Konvertierung Ungenauigkeiten richtige Darstellung
<code>\num{1E-7}</code>	1 × 10 <sup>-7</sup>	Konvertierung wissenschaftliche Notation

### 3.2.5. Abkürzungen

Bei Abkürzungen wie “z. B.” oder “i. d. R.” ist auf richtigen Abstand zu achten. Falsch ist es, keinen Abstand (“z.B.”) oder einen zu großen Abstand (“z. B.”) zu wählen. Zudem muss verhindert werden, dass derartige Abkürzungen durch einen Zeilenumbruch auseinander gerissen werden. Aus diesem Grund wird ein so genanntes “Spatium” verwendet, dass ein schmales Leerzeichen darstellt, welches nicht umbrochen werden kann. Das Spatium wird in *TEX* mit der Zeichenfolge `\,` erzeugt, siehe [Table 3.6](#). Für häufig wiederkehrende Abkürzungen ist es sinnvoll Makros zu definieren. Diese können in dieser Vorlage in der Präambel der *main.tex* definiert werden.

**Table 3.6:** Abstände bei Abkürzungen .

richtig		falsch	
Code	Darstellung	Code	Darstellung
<code>i.\,d.\,R.</code>	i. d. R.	<code>i.d.R.</code>	i.d.R.
<code>z.\,B.</code>	z. B.	<code>z. B.</code>	z. B.

## 4. Base Features

Let us first go through some of the base features offered by  $\LaTeX$  and used in this template. Before anything happens, the  $\TeX$  engine is chosen. This is the binary responsible for compiling the  $\LaTeX$  source code. If you have used  $\LaTeX$  before but never heard those terms before, you are most likely using  $\text{pdf}\LaTeX$  as your engine. This is the go-to default for generating PDF output. Its most popular modern alternatives are  $\text{Xe}\LaTeX$  and  $\text{Lua}\LaTeX$ . These two offer various new, shiny features, chief among which is Unicode support through the **unicode-math** package, which builds onto **fontspec** and extends it with math fonts capabilities. Note that these packages *require*  $\text{Xe}\LaTeX$  or  $\text{Lua}\LaTeX$  use:  $\text{pdf}\LaTeX$  is *old* and will no longer work. More on that in [Section 4.1](#).

$\text{Lua}\LaTeX$  is preferred over  $\text{Xe}\LaTeX$  for the following reasons:

1. **contour**, a package that can do things like ~~this~~ works. For an application of that, see [Figure 5.6](#) on [Page 69](#). Getting **contour** to work on  $\text{Xe}\LaTeX$  is somewhat impossible, see also [here](#), [here](#), [here](#) and [here](#).
2.  $\text{Lua}\LaTeX$  allocates as much memory as it happens to need. The ancient limitations of  $\TeX$  are easily reached by packages like **tikz** and **pgfplots**. Circumventing that either feels hacky, or actually is hacky. `tikzexternalize` is a great module, but has some caveats, like breakage of clickable references. It solves a problem that should no longer exist in the first place.
3. The fantastic **microtype** package has a number of unavailable functionalities when using  $\text{Xe}\LaTeX$ .

Using Lua $\text{\LaTeX}$ , you do not have to worry about any of that: hit compile, wait, done<sup>1</sup>.

## 4.1. Fonts and Text

Using high-quality fonts is one goal. This includes the modern and elegant [Libertinus fonts](#), of which the *Libertinus Serif* was chosen for this document. It comes with an accompanying [math font](#) of the same name<sup>2</sup>. There are not very many high-quality free fonts with a math companion available, see also [this compilation](#). A similar compilation of “nice” fonts is [found here](#), however that is specifically for pdf $\text{\LaTeX}$ , not Lua $\text{\LaTeX}$ . The font choice can be changed with relative ease in the options for the `unicode-math` package. If a new font does not provide at least the same array of features, parts of the document might break! For reference, there is a list of [symbols defined by unicode-math](#). Using both normal and maths fonts in conjunction is made possible through the `unicode-math` package. As such, an exact match of the text and math fonts is achieved. This is important but often overlooked. However, such a match is often plain unavailable in the typesetting tool at hand.  $\text{\LaTeX}$  is no exception here if there simply exist no matching fonts. We took care to ensure a complete match here.

Not only do the fonts look great on their own and paired up, they (just as importantly) also feature very broad support for all sorts of symbols and characters, as well as font shapes and weights and combinations thereof. The latter is demonstrated in [Table 4.1a](#). Combine all that with `microtype` (a package taking care of typesetting details), and we get very beautiful typesetting. For example:

*Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies*

---

<sup>1</sup>The *waiting* part might be a bit longer than what you might be used to from pdf $\text{\LaTeX}$ . So far, this is the only disadvantage I could find.

<sup>2</sup>Both fonts are vector fonts; if  $\text{\LaTeX}$  yields any warnings about *font size substitutions*, that is bogus and can be ignored.



Feature	Sample Text
Regular	The quick brown Fox jumps over the lazy Dog 13 times!
<b>Bold</b>	<b>The quick brown Fox jumps over the lazy Dog 13 times!</b>
<i>Italics</i>	<i>The quick brown Fox jumps over the lazy Dog 13 times!</i>
<b><i>Bold Italics</i></b>	<b><i>The quick brown Fox jumps over the lazy Dog 13 times!</i></b>
SMALL CAPITALS	THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG 13 TIMES!
<b>BOLD SC</b>	<b>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG 13 TIMES!</b>
<i>ITALICS SC</i>	<i>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG 13 TIMES!</i>

**Table 4.1a:** Examples for font features offered by the main roman font. Notice the small vertical white space after the fourth row, nicely separating content that is slightly dissimilar.

*vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.*

Notice the balanced line endings and low number of hyphenation; easy on the eyes and very readable.

**Old approach** The overwhelming majority of  $\text{\LaTeX}$  documents relies on the two lines

```
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
```

This approach is **outdated** and only required when using pdf $\text{\LaTeX}$ . Since this is what most people still do, the two packages are still required. However, these two packages should at least not be used for *new* work anymore! Using Lua $\text{\LaTeX}$ , they are not required anymore:

- input encoding is automatically UTF-8 (as it should be in  $\$CURRENTYEAR$ ),

- font encoding is also not required, since fully capable fonts are used, which come with all the glyphs required.

The same is true for  $\text{\LaTeX}$ .

### 4.1.1. Math

What often does not occur at first glance is that many documents use text and math fonts that are very different from one another. As far as I know, Microsoft's *Word* has usable math typesetting and sensible default fonts for that. Yet, it cannot do what dedicated, fully-fleshed text and math fonts — different, but matched — can achieve. They provide a seamless transition, especially when using actual text in the math environment or vice versa, like when we go for  $x \rightarrow \infty$  and then also do  $\int_1^2 y^2 dy$  or maybe  $a^2 + b^2 = c^2$ . All these inline math elements look perfectly natural. If instead the fonts did not match, inline math would stick out like a sore thumb. Toggle the colors in the class file (`*.cls`) to highlight each different font family to see all the differences (see the `Color` option for `unicode-math`). Some display-style math examples follow.

$$pv = RT \quad [4.1]$$

$$e^{i\pi} + 1 = 0 \quad [4.2]$$

$$\lim_{x \rightarrow \infty} \frac{\pi(x)}{x / \ln(x)} = 1 \quad [4.3]$$

$$\sum_{k=0}^n \binom{k}{n} = 2^k \quad [4.4]$$

$$\left[ T \frac{\partial}{\partial T} + \kappa(p) \frac{\partial}{\partial p} + \alpha \right] \vartheta^k(x_1, x_2, \dots, x_k) = 0 \quad [4.5]$$

$$\Delta h_{\text{change}} = 1/2(c_{\text{exit}}^2 - c_{\text{entry}}^2) \quad [4.6]$$

If these symbols are colored, it is because they are links and link coloring is on. This can be turned off (globally or for certain elements) in the options to `hyperref`. If color is turned off, they are still hyperlinks. If even that is undesired, that can also be turned off in that package.

## Predefined Macros

The `physics` package has some `issues`, and as such, this document relies on a couple custom macros in place of it, see Table 4.2.

## Symbols

Note how the symbols in equations are hyperlinks (leading to their definition in the glossary), courtesy of packages `hyperref` and the powerful `glossaries-extra`. Having not specified anything else, they take on whatever hyperlink color was specified in the preamble. In the final document, all links should be hidden, aka black. This is a given for print output, but probably also sensible for digital output. The visual noise introduced by colored links is quite immense. Their only point is to let users know there is something clickable — there is a good chance that is figured out anyway. If at all, use a dark green or blue tone.

## Math Highlighting

In a very unobtrusive yet also unambiguous way, we can highlight important results, as shown in Equation 4.3. This feature is a natural part of `tcolorbox`, a very powerful package for anything color and boxes. Note that for print, the gray tones should probably be darkened.

Name	Examples
Mean	$\bar{\rho}, \bar{A}$
Logarithmic Mean	$\tilde{\rho}, \tilde{A}$
Absolute <sup>1</sup>	$ \rho ,  A , \left  \frac{A^2}{A^3} \right $
Flow	$\dot{m}, \dot{h}$
Delta	$\Delta V, \Delta h$
Nabla Operator	$\nabla x, \nabla^3 x$
Vectors	$\mathbf{x}, \mathbf{A}$
Derivatives	$dm, d^2x, \partial m, \partial^2 x$
Fractional Deriv.	$\frac{dU_{el.}}{dt}, \frac{d^2y}{dx^2}, \frac{\partial U_{el.}}{\partial t}, \frac{\partial^2 y}{\partial x^2}, \frac{dh}{dT}, \frac{d^2h}{dT^2}, \frac{\partial h}{\partial T}, \frac{\partial^2 h}{\partial T^2}$
Time Deriv. <sup>2</sup>	$\frac{dU_{el.}}{dt}, \frac{d^2y}{dt^2}, \frac{\partial U_{el.}}{\partial t}, \frac{\partial^2 x}{\partial t^2}$
Positional Deriv. <sup>2</sup>	$\frac{dU_{el.}}{dx}, \frac{d^2y}{dx^2}, \frac{\partial U_{el.}}{\partial x}, \frac{\partial^2 y}{\partial x^2}$

**Table 4.2:** Predefined math macros. Refer to the source code for help on the syntax. For derivatives, the starred macros yield partial derivatives.

## Indices and Operators

Indices are typeset upright! They are text. Math output like  $x_{upper}$  is one of the most common mistakes. If text occurs in math mode, that is also actual text. Consider

$$MEAN_{sample} \neq 2, \quad [4.7]$$

which looks stupid. What Equation 4.7 really says is “ $M \cdot E \cdot A \cdot N$ ”: slanted characters are variables, and not specifying any operator implies multiplication. This example is pedantic, but it is very easy to imagine such an issue evolving into real ambiguity, which would then be a serious error. Since multiplication is implied by doing nothing, leaving out `*` aka `\cdot` altogether often looks best.

Use `\DeclareMathOperator` to define custom operators in upright font, then use `\text` for the subscript:

$$MEAN_{\text{sample}} \neq 2. \quad [4.8]$$

## Text-Flow

Equation 4.8 is part of the surrounding sentence. Math is just another written “language” (the only one understood around the globe!) which can and will be read as a natural part of the surrounding text. As such, it should contain punctuation marks. So that now, having considered that we have the amazing result of

$$1 \neq 2, \quad [4.9]$$

we have achieved better overall reading flow, with commas where there would naturally be separations when reading the sentence out loud as a whole. Notice the small spaces `\,` before the commas and dots in equations. They are convenient to avoid ambiguities. Implementing these as `\eqcomma` and `\eqend` ensures consistency. Additionally, these punctuations can then also be switched off globally if your requirements demand so. A second example often occurs when symbols are defined. Notice the difference between these:

1. The density  $\rho$  is defined in Equation 4.10.

$$\rho := m/V \quad [4.10]$$

2. We can therefore define the density as

$$\rho := m/V. \quad [4.11]$$

The example in Item 2<sup>3</sup> reads more naturally, is less clunky, there is less duplication of code and information and the reader does not have to jump around references. Nevertheless, embedding equations into the surrounding text is unfortunately somewhat subjective.

## Physical Units

In the name of all that is holy, *never* manually type out units, numbers or quantities yourself: use `siunitx`. The package is an absolute must-have if there are any units to be typeset. It is also worth using if there are no units, but numerals. The latter can be typeset using the `\num` command. It is capable of parsing number constructs:

<code>\num{1.0}</code>	$\rightarrow 1.0$	
<code>\num{1,0}</code>	$\rightarrow 1.0$	(Localisation)
<code>\num{1.0}</code>	$\rightarrow 1.0$	(German)
<code>\num{1.2e3}</code>	$\rightarrow 1.2 \times 10^3$	
<code>\num{1.2e-7}</code>	$\rightarrow 1.2 \times 10^{-7}$	
<code>\num{-e7}</code>	$\rightarrow -10^7$	
<code>\num{3.2(9)}</code>	$\rightarrow 3.2 \pm 0.9$	(Uncertainty)
<code>\num{300000000}</code>	$\rightarrow 300\,000\,000$	(Group separation)

---

<sup>3</sup>Items in lists can be labeled and referenced using `enumitem` and `cleveref`.

If the roman text font uses hanging numerals (like: 1234567890), but a stand-alone number needs to be typeset, `\num` is also required. In this document, it will use the math font: 1 234 567 890.

Physical units are output using `\unit`. It also has parsing capabilities:

```
\unit{\meter\cubed\kelvin\per\kilogram\squared\per\giga\watt\per\degreeCelsius}
```

will print  $\text{m}^3 \text{K}/(\text{kg}^2 \text{GW}^\circ\text{C})$ . The mode in which units with negative exponents are displayed can be changed, e.g. fractions can be used:  $\frac{\text{m}^3 \text{K}}{\text{kg}^2 \text{GW}^\circ\text{C}}$ . You can also use `\DeclareSIQualifier` to create textual subscripts for units, further specifying them, or create entirely custom units using `\DeclareSIUnit`. An example for both combined is:  $\frac{\text{MWh}_{\text{th}}}{\text{m}^3}$ . The `\qty{<quantity>}{<unit>}` command essentially combines `\num` and `\unit`, inserting a small space in between, for correct typesetting. No or full spaces are wrong and painful to read.

There are also commands to print ranges: 20 to 50 m. Refer to the package documentation for more.

## Chemistry

Closely related to purely mathematical equations are chemical ones, as shown in [Equation 4.12](#).



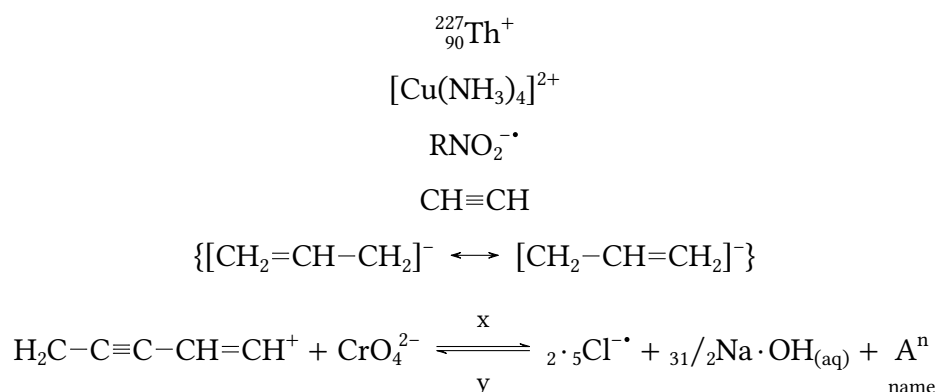
Aligned arrays of multiple chemical reaction equations are also possible, as demonstrated in [Equations 4.13](#) and [4.14](#).



Single chemical compounds are invoked using `\chcpd`, as demonstrated in [Equation 4.15](#).

$$\frac{H_i}{1 \times 10^6} = 35\gamma_C + 94.3\gamma_H + 10.4\gamma_S + 6.3\gamma_N - 10.8\gamma_O - 2.44\gamma_w \quad [4.15]$$

These commands are provided by `chemmacros` and its `chemformula` module. Those packages are very capable, so more complex chemistry can also be typeset:



All these examples are courtesy of the documentations of these two packages.

### 4.1.2. Sans-Serif

Having taken care of the main, aka roman, font, the sans-serif font is the logical next step. [Fontin Sans](#) is a decent such font. Importantly, and in contrast to most other free sans-serif fonts, it comes with all the bells and whistles required for stunts, see [Table 4.2a](#). This even includes small-capitals support.

### 4.1.3. Mono-Spaced

Wanting to display any sort of code, or maybe a Uniform Resource Locator (URL), in the document will have you looking for a mono-spaced aka typewriter font. Modern [Monospace Neon](#) from the Monospace family provides superior monospaced typography, replac-



Feature	Sample Text
Regular	The quick brown Fox jumps over the lazy Dog 13 times!
<b>Bold</b>	<b>The quick brown Fox jumps over the lazy Dog 13 times!</b>
<i>Italics</i>	<i>The quick brown Fox jumps over the lazy Dog 13 times!</i>
<b><i>Bold Italics</i></b>	<b><i>The quick brown Fox jumps over the lazy Dog 13 times!</i></b>
SC	The quick brown Fox jumps over the lazy Dog 13 times!

**Table 4.2a:** Examples for font features offered by the sans-serif font.

ing the previous Inconsolata choice. See [Chapter 6](#) for a more in-depth demonstration. At the time of writing (April 15, 2020)<sup>4</sup>, Monospace Neon has native regular, bold, and italics faces, providing complete typographic support without any font faking required. This represents a significant improvement over the previous monospace font that required artificial slant for italic appearance. Italics are particularly useful for code listings with comments and documentation, so native support greatly enhances readability. See for yourself in [Table 4.2b](#).

#### 4.1.4. Symbols

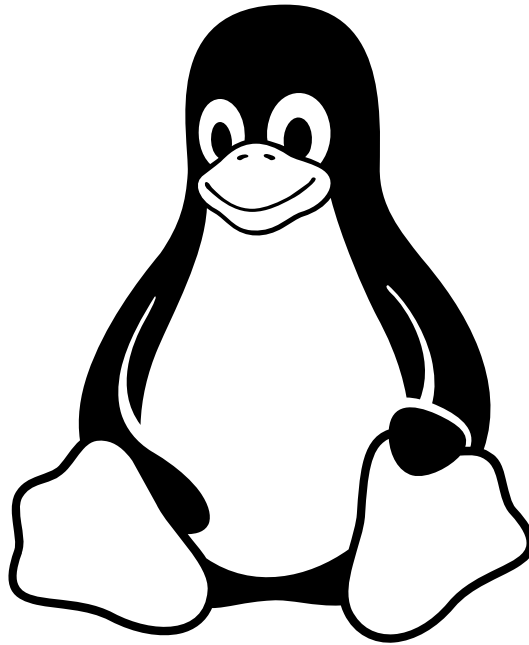
**fontawesome5** is a package providing hundreds of freely usable vector symbols. They scale just like normal vector graphics and can be used alongside text, like this little friend

<sup>4</sup>Date typeset as \DTMdate{YYYY-MM-DD} using **datetime2**. This gets us **language-independent** International Organization for Standardization (ISO) formatting.

Feature	Sample Text
Regular	The quick brown Fox jumps over the lazy Dog 13 times!
<b>Bold</b>	<b>The quick brown Fox jumps over the lazy Dog 13 times!</b>
<i>Italics</i>	<i>The quick brown Fox jumps over the lazy Dog 13 times!</i>

**Table 4.2b:** Examples for font features offered by the monospaced font.

here: 🐧. They can also be colored (🐧) or scaled:



All of these should also be select- and then copyable, yielding e.g. *LINUX*. Refer to the package documentation for a list of all the available symbols.

## 4.2. Language

Closely related to the previous section are the language features. Currently, there are two supported languages: English and German. See for yourself in the file responsible for the translations, `translations.tex`. It leverages the `\providecaptionname{}` command from KOMA-Script. Given the name of that command, it is probably a misuse and not its purpose, but it worked better than its alternatives.

The **polyglossia** package does all the heavy lifting for this document. It is an alternative to the widespread **babel** package. Note that **polyglossia** requires the language to be passed to its `\setdefaultlanguage{}` command. It will normally not work by just specifying a global document language. However, you can pass a global language to the

language= keyword in the options to `\documentclass{acp}`, and it will be picked up correctly.

The chosen language option will also be forwarded to various other packages, like **enquote** for quotations and **siunitx** for localization of numerals typesetting. **polyglossia** will take care of all the hyphenation rules, so there should be no or very minimal need for manual `\hyphenation{}` or `\-` commands.

The language can be switched dynamically, and all content will be adjusted, see [Table 4.3](#). More languages can be added easily in the translations file.

## 4.3. Sectioning

### 4.3.1. Explanation

Between any two sectional commands (from `\part` down), there always should be at least *some* content. Anything else looks off, *cf.* the direct transition between [Sections 4.3](#) and [4.3.1](#). At least, tell the reader what the following hierarchical level contains for them. Further, three numbered levels of hierarchy are enough. The numbering and even availability of sectioning commands depends on the used `documentclass`. In **koma-script** articles (`scrartcl`), these are sections, subsections and subsubsections. **koma-script** reports (`scrreprt`) and books (`scrreprt`) extend this by parts and chapters, both of which numbered. Subsubsections will no longer be numbered. Trying to create a `\part` or `\chapter` in article classes will result in errors.

**Table 4.3:** Available languages and dynamic switching. Note how the commands are language-dependent; no manual adjustments necessary.

Language	Quotes	Numerals	Headers <i>etc.</i>	Hyphenation
English	“Quote!”	100.2 m	HEADER	
German	„Zitat!“	100.2 m	ÜBER- ÉR- SCHRIFT	

## Deeper!

Deeper sectioning is fine, but it should not be numbered or occur in the table of contents. Notice how, without specifying anything, **koma-script** abides by that automatically. This is despite using `\subsubsection{Deeper!}` as opposed to `\subsubsection*`, its explicitly unnumbered counterpart.

Also notice how between `\section{Sectioning}`, which produced [Section 4.3](#), and `\subsubsection{Deeper!}`, there was no subsection. Usually, you would want a clean cascade with no jump in the hierarchy levels. If you catch yourself jumping, it is also a sign there might be something off with your actual content structure or thought process. That being said, it is absolutely okay to do; just be aware.

$\text{\LaTeX}$  has many ways where it does not let you do something easily, or where commands look off. *Never fight  $\text{\LaTeX}$*  in that; you will lose, it will win. When something is awkward to do or requires a lot of manual labor, you are probably doing it wrong and there will be an easier way. Most often, that comes in the form of packages. Being a fully-fledged programming language under the hood, *any* repetition in  $\text{\LaTeX}$  should make you stop for a second and wonder about another way. That may lead to you discovering suboptimal approaches and code. Just as often, it will not lead anywhere and trying to force minimum code and maximum optimization will not be economical. After all,  $\text{\LaTeX}$  is a markup language.

**Paragraphs** They are a nice touch, introducing a new, distinct paragraph, idea or thought without being too loud about it.

That being said, leave a blank line in the source code for regular paragraphs. Use either vertical spacing between paragraphs or indentation (like here). Do not use both! Each new train of thought should go into its own paragraph. This paragraph is not indented, as was intended by manually calling `\noindent`. In regular text and documents, there are very few reasons that should ever occur. Again, let  $\text{\LaTeX}$  do its thing; if you find yourself typing `\noindent` all the time, there will be something wrong. For example, paragraph styles (vertical spacing between them, indentation *etc.*) is configured globally.

## 4.4. References

This **section** is about referencing. Notice `\lcnameref` in the previous sentence: it references (in lower-case) the name of the passed label. If this section ever changes to something else (chapter, ...), it is updated automatically.

Note that using package **cleveref**, we only ever issue `\cref{<label>}` commands.

That command also has many useful cousins, like `\crefname{<label>}`. The package does the heavy lifting and inserts the reference *type*, also with correct plural forms if required: [Table 4.1a](#) and [Figures 4.1](#) and [5.6](#) ← all of that was done automagically. Doing it any other way is just way too laborious and error-prone.

For added convenience, insert the *type* directly into the label, e.g. `\label{fig:hello}`. This aids auto-completion and readability.

### 4.4.1. Bibliography

The second place where references occur are bibliographical contexts. Examples are spread throughout this document. In this document, the use of citations is facilitated by **biblatex** and its backend **biber**, with **bibtex** considered outdated. **biblatex** is recommended for its UTF-8 support, essential for citing authors with international names. For reliable source citation without worrying about specific citation styles (chicago, apa, etc.), use the command `\cite {bibid}`. This command adapts globally to the desired citation style and is correctly defined for your thesis. For detailed examples of various citation commands and examples, see [Table 4.4](#).

Citations can be placed in the text in various ways:

- Directly in the text with `\textcite {bibid}`
- In brackets at the end of a sentence or paragraph with `\cite {bibid}`
- To cite multiple sources in one citation, `\cites {}`, `\textcites {}`, `\parencites {}` can be used
- Page numbers or notes (if this note is an integer number, it is automatically taken

to be the page number) can be directly added to each citation with `[]`. And adding `\psq` for the following page or `\psqq` for the mentioned and subsequent pages is also possible in the notes (and language dependent!).

As such, we can have citations looking like:

- [EINSTEIN 1905]
- [DIRAC 1981; EINSTEIN 1905]
- [EINSTEIN 1905, p. 8; GOOSSENS et al. 1993, pp. 29 sqq.; Donald Ervin KNUTH 1986, pp. 2–9; Donald E. KNUTH 1973, pp. 89 sq.; DIRAC 1981].
- (BAEHR and KABELAC 2016; BIRD et al. 2009) or (vgl. BAEHR and KABELAC 2016, pp. 15 sq.; BIRD et al. 2009, pp. 132 sqq.)
- BAEHR and KABELAC (2016, p. 5) and DUBBEL et al. (2007, pp. 50–56)

The documentation offers a comprehensive overview of the wide range of automatic citation commands and should be consulted for a deeper exploration.

**Table 4.4** / Overview of Citation Commands and usage examples.

Command	Output	Example
<code>\cite {einstein1905}</code>	[EINSTEIN 1905]	The theory of relativity deals with the structure of space and time and the nature of gravitation. [EINSTEIN 1905]
<code>\textcite {einstein1905}</code>	EINSTEIN (1905)	EINSTEIN (1905) demonstrated that time is relative, revolutionizing physics.
<code>\parencite {einstein1905}</code>	(EINSTEIN 1905)	The theory of relativity ((EINSTEIN 1905)) changed our understanding of space and time.
<code>\textcquote {einstein1905}{quote}</code>	“quote” (EINSTEIN 1905)	Einstein remarked: “Mass and energy are equivalent.” (EINSTEIN 1905).
<code>\enquote {\ldots }</code>	“...”	Einstein’s principle that “God does not play dice” emphasizes his rejection of quantum mechanics.



**Language support** All of this is incredibly convenient, for there is minimal manual work and a lot of abstraction into high-level commands. Importantly, this is language-agnostic, and **biblatex** will make use of **polyglossia** (the Lua $\text{\LaTeX}$  replacement for **babel**) and **csquotes**. Therefore, through changing the desired document language for just these packages, all others including **biblatex** will quickly adjust automatically.

**backref-feature** Taking a look at the bibliography (Page 115) in the backmatter of this document reveals the **backref** feature: the pages where a reference was cited occur after its entry. This is helpful in print, but amazing in digital format, for these page numbers are also links. This allows readers to very swiftly navigate and jump within the document. Try it out by following this reference: [DIRAC 1981], leading you to the bibliography. It will have this page's number (Page 49) at the end of its entry. Clicking it will land you back here exactly.

**Citation Manager** To generate the \*.bib file from which  $\text{\LaTeX}$  pulls the info in the first place, a citation manager is a must-have. Zotero (especially with its excellent *Better Bibtex* addon) is recommendable, but it ultimately does not matter much. Just use *some* manager to keep your actual documents (consisting of bibliographical info and the attachments themselves, like e-books) and the automatically derived \*.bib files in one place, named and structured consistently.

## 4.5. Lists

In technical publications, using lists (either bullet points or enumerations) is highly encouraged. They should almost always be preferred over doing the same thing in a block of text. Just consider this example list:

- The demonstrated approach is more complex than the previous one:
  1. more time was spent doing computations,
  2. less was spent fooling around,

3. features were added.

- At the same time, the following simplifications were made:

1. went from continuous- to discrete-time simulations,
2. threw out some superfluous stuff.

The very same information in form of a text block is suddenly much more inaccessible to readers. In technical writing, precision and conciseness matter — prose, synonyms and filler words do not.

Note the block-like `itemize` symbol (☒)<sup>5</sup> and the fact that enumerate numbers are part of the sans-serif family and **bold**. That is totally revolutionary and stuff, because it's... different? If it is not to your liking, lists may be changed, configured and created using the `enumitem` package. For example, using that package, this document has a list for numbered descriptions, in essence a combination of the `enumerate` and `description` environments:

1. **This item** has this description.

2. **This other item** has a different description.

## 4.6. Censoring

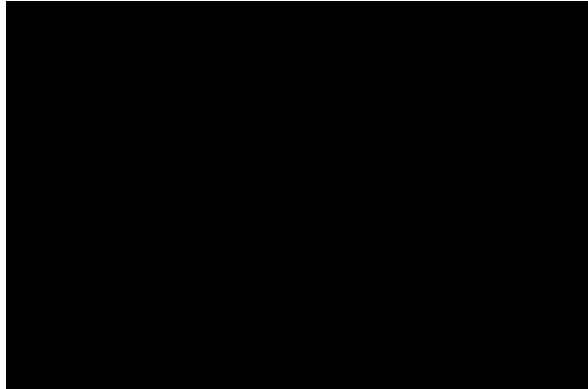
There is a censoring feature, provided by `censor`. It allows you to publish documents containing sensitive information, *e.g.* for proofreading. So, this is now a huge secret:

████████████████████

Float contents can also be censored, as illustrated in [Figure 4.1](#).

<sup>5</sup>This is also a regular Unicode character. When copy-pasting old or outdated  $\LaTeX$  documents, ever noticed all the funny business going on? A good example is copying German *umlauts* like Ä, Ö, Ü. They might turn into ¨A, since  $\LaTeX$  only ever put two random dots over the regular ASCII letter A. With `unicode-math` (building onto `fontspec`) and Lua $\LaTeX$ , this document is fully Unicode-compatible. Copy-paste anything correctly:  $\delta\sigma\int_1^2\gamma$ . Googling the ☒ block will reveal that it is U+25AA.

I am a  
TODO note.



**Figure 4.1:** Example for a censoring box.

## 4.7. Glossaries

**glossaries-extra** is an absolute unit of a package. For this document, it is used with its back-end **bib2gls**. This Java tool comes with installations of TeXLive and MiKTeX. You should already have it. This section is only a brief, opinionated overview. For a more appropriate introduction, there is a suitable [Beginner's Guide](#), written by the package author. Before you jump further into the relevant, proper documentations (these are three: the **glossaries** base package, the comprehensive extension **glossaries-extra** and the helper tool **bib2gls**), you might want to give the beginner's guide a shot. In a format similar to normal bibliographies, all glossary entries are now managed using `*.bib` files. Each entry is processed by **bib2gls** and put into an auxiliary file. From it, **glossaries-extra** reads and inserts all contents when `\gls` and its many sibling commands are used. The framework for all of that, also printing of the glossary and nomenclature, is already taken care of for this document. In addition to printing the glossaries, they are also added to the table of contents. This behavior can be toggled using the `toc` package option flag, see the source code for more info.

Traditionally, glossary packages are used for abbreviations. However, using **glossaries-extra** this can be kicked into top gear. It is used for:

- abbreviations,

- (physical) constants,
- symbols (greek, roman and all other),
- subscripts,
- the book index, consisting of names and the normal index.

In the case of symbols, this means the source now relies on `\sym{<label>}` commands, see also Table 4.5. For example, equations no longer read  $E = mc^2$ , but instead

$$\text{\sym{energy}} = \text{\sym{mass}}\text{\sym{velocity}}^2$$

This initially unintuitive approach has several critical advantages:

1. absolute **consistency** following the Single Source of Truth (SSOT) principle: there is exactly *one* place where the symbol itself is defined. All other usages are just *references* to this central definition. If suddenly, the symbol for velocity has to change from  $c$  to  $v$  throughout the entire document, it can be done with ease. Replacing single letters like that using tools like `sed` would be a nightmare, if not impossible. There is also absolutely no danger that *both* symbols occur (unless it is explicitly set up like that), with both referring to velocity, as can happen when returning to a document after abandoning it for many months or if multiple authors work on one document.
2. following the **What You See Is What You Mean (WYSIWYM)** principle: this is  $\text{\LaTeX}$ 's big strength. In the source code, it no longer states what you *want to see*, but instead *what you mean*. When you write  $E = mc^2$ , you are writing what you want to see: the letter  $E$  is somehow equal to the product of letter  $m$  times  $c$  squared. But the *meaning* is that *energy* equals *mass* times *velocity* squared. This can now be expressed directly in the source code.

The WYSIWYM principle is at the core of  $\text{\LaTeX}$  and is what sets it apart. It is the reason you also do not say

```
{\Huge\textsf{\textbf{<Section Title>}}}
```

but instead simply

```
\section{<Section Title>}
```

In the first, it was stated what the author wants to *see*, but only in the second one does it say what was *meant*. It is painfully obvious why the latter approach is the correct one. Another example is emphasized text. Do *emphasized* (produced from `\emph`) and *emphasized* (`\textit`) text look the same? They certainly do... usually.

Yet, what is *meant* is *emphasis*; italic text is just what it happens to look like now, but it is not the *meaning*. For example, we could later decide to redefine emphasized text to bold, or colored. If you previously did not differentiate strictly enough between `\emph` and `\textit`, you are in for a bad time. This is a trap beginners unfortunately often fall into. Using **glossaries-extra**, abstracted markup-commands can be taken to a whole next level, leveraging this core  $\text{\LaTeX}$  strength.

3. **abolishing ambiguity**: especially when the source code is read by other people, or even worked on, “naked” symbols can become ambiguous. What American authors refer to with a capital *P*, European authors interpret as *power*, when really *pressure* was meant. This is a non-issue if in the source it says `\sym{pressure}`. For internationalization, authors would then only adjust their *style-sheets* (in this case the `*.bib` files) and be done.
4. arguably improving **readability**. The source code can almost be read like a normal sentence consisting of full words, albeit with braces and backslashes in the way.
5. printing the **nomenclature** becomes trivial. A single command simply prints the `*.bib` file contents. Being an external Java tool, the customization, sorting and filtering capabilities of **bib2gls** for printing those lists are likely more than will ever be needed.

6. lastly, some more gimmicky features are enabled, like printing all page numbers of occurrences of a symbol. Alternatively, only the first occurrence can be printed.

**Available Commands** `glossaries-extra` is already heavily leveraged for this document. Take a look around the source code for all the details. For starters, there are a couple of predefined `*.bib` files, showcased in [Table 4.5](#). Note that using entries with commands like `\sym{<entry name>}` is independent of the entry type used in the `*.bib` file. For example, subscripts are invoked using `\sub{<entry name>}` despite being defined as `@symbol{<entry name> ... in subscripts.bib}`.

**Abbreviations** are displayed in the long form, when they are first used, with the short form in brackets. After that, they are always used in the short form. In this first call Comprehensive T<sub>E</sub>X Archive Network (CTAN) is written out. Now, in it's second call the short form is used: CTAN. Although this is very useful, you sometimes might want to access the long form, although it's not the first time. E.g. in a new chapter or whatever. The command therefore is:

```
\glstrlong{<category>.<entry name>}
```

Category is here the same as the command mentioned in [Table 4.5](#) without the backslash. Like this we can use the long form again: Comprehensive T<sub>E</sub>X Archive Network. If you want to reset all your abbreviations in a chapter use the command `\glsreset` in the beginning of your chapter and all abbreviations will be shown in the long form again for the “first” time. Or to just reset a specific abbreviation use `\glsreset{<category>.<entry name>}`.

Another useful command might be, to access the description, aka the name of your variable, for symbols. Therefore just use `\glsdesc{<category>.<entry name>}` like here Area for the Symbol A.

---

<sup>1</sup>Constants like  $\pi$  are toy examples for this document. However, this glossary section is very convenient to share assumptions and used constants in a clear and concise way in one central place, aiding reproducibility and overall document integrity.

**Table 4.5:** Predefined glossaries with their respective \*.bib files, invoking commands and list occurrence in the document.

Name (.bib)	Command	Listed in / Description
terms	\idx	Index → Terms
names	\name	Index → Names
roman	\sym	Glossary → Symbols → Roman
greek	\sym	Glossary → Symbols → Greek
other	\sym	Glossary → Symbols → Other
subscripts	\sub	Glossary → Subscripts
constants <sup>1</sup>	\cons	Glossary → Numbers
abbreviations	\abb	Glossary → Acronyms

**Modifying Print Output** To use glossary entries but not print them in a glossary, comment out the entire relevant `\printunsrtglossary` macro where appropriate. For the index, the relevant command is `\printunsrtindex`. Some glossary categories, like *Symbols*, have sub-categories, like *Roman*, *Greek* and *Other*. To omit printing a single sub-category in the glossaries (but still allow their use in the document<sup>6</sup>), refer to the `notprinted` type and the instructions found in the \*.cls file.

**German Umlauts** As you can see, even glossary entries with german umlauts work: Wärmepumpe (WP). And if it is used the second time (like here: WP) the short form is used. This feature is achieved by the magic comment in the head of every .bib file of the glossary:

```
% Magic Comment: this template requires UTF-8 encoding. The following 'magic
↪ comment' is
% recognized by bib2gls and will ensure UTF-8 is used:
% Encoding: UTF-8
```

<sup>6</sup>The *Other* symbol entries are required for the provided built-in math macros, see Table 4.2, to work!

% see: <https://github.com/nlct/bib2gls/issues/5#issuecomment-490863175>

### 4.7.1. bib2gls

**bib2gls** is an external tool (of the same name as the package) that **glossaries-extra** employs to convert external `*.bib` files with definitions for, for example, acronyms, into a  $\text{\TeX}$ -compatible format. During conversion, it also processes all the entries, like sorting them in whatever way you request. The sorting and filtering capabilities are very strong, and of course also Unicode compatible. A minimal `bib` file for acronyms can be as simple as:

```
@abbreviation{cont_int,
  short={CI},
  long={Continuous Integration},
}
```

For concrete examples, see the files for this document at `bib/glossaries/`. There can be as many keys as required, with custom ones being easily created. Effectively, this is analogous to how bibliography entries are created. Thus, users of  $\text{\LaTeX}$  who are already familiar with that concept and format should have an easy time getting started with **glossaries-extra**. Using it for mathematical symbols can look like:

```
@symbol{abs_temperature,
  name={\ensuremath{T}},
  description={absolute temperature},
  group={roman},
  unit={\unit{\kelvin}},
}
```



## 4.8. Landscape

Pages in landscape format are rather straightforward to implement. Note that not only are these in landscape orientation; they are also recognized as such by supporting document viewers, rotating the page for you and keeping it legible.

Rest of this page intentionally left blank.

## 5. Float Features

Examples for floats were already shown in [Figure 4.1](#) and [Table 4.1a](#). These are mainly handled by the packages:

**caption** The base package providing customizations for the caption text, for example automatic ending periods, which can be toggled globally.

**svg** Including Scalable Vectors Graphics (SVG) files using  $\LaTeX$ , or in this case Lua $\LaTeX$ , is not very straightforward. In fact, it is anything but: SVG files cannot be included directly at all and intermediate steps are needed.

Using the **svg** package, the workflow is somewhat automated. Only the original SVG files are kept as the SSOT, and the generation of the PDF and accompanying `.pdf_tex` files are left to the package. It calls Inkscape for converting the SVG to PDF (or another format of choice), and if the SVG contains text to be included as  $\LaTeX$ , a sidecar `.pdf_tex` file is generated (the default behavior). To call Inkscape, two requirements have to be met:

1. elevation aka `--shell-escape` is required to write external files, and
2. Inkscape has to be on the `$PATH`, that means installed and subsequently registered. This is automatically done for Linux. If it is not done automatically on Windows, navigate to *Edit the system environment variables* and add the directory containing `inkscape.exe` to the variable.

Once the files are generated, they can be treated as temporary junk and are always easily regenerated from the source SVG.

After years of experimentation, this seems like the best workflow. The only laborious manual task left is placement of annotations onto the generated PDF files. This seems like the best deal: no text is left in the SVG files themselves. Placing and debugging text in SVG files using the Inkscape  $\rightarrow$  PDF+PDF\_TEX route is *very, very* annoying. This is because while Inkscape offers text alignment operations (left, center, right) that translate into the embedded PDF\_TEX, the font cannot be known a priori while working on the SVG. Neither font size (most importantly its height), nor any other font property can be assumed. This also makes functions like *Resize page to drawing or selection* futile if text is part of the outer elements of a drawing.

Wanting to change any text later on results in having to start Inkscape instead of just doing it conveniently in the  $\text{\LaTeX}$  source. The alternative is to place macros (`\newcommand*{}`) everywhere inside the original SVG, where content should later be placed. These macros serve as labels, but are ugly, annoying, and remove the usability of the plain, original SVG file (since we would first need to know what each macro stands for).

Using the `svg` package to generate plain, text-less PDF and only later adding any text or annotation in the  $\text{\LaTeX}$  source itself seems the best of both worlds. **It certainly allows both tools to do what they're good at, and no more:** draw free-flowing vectors graphics with Inkscape, then add text in  $\text{\LaTeX}$  (which can be done in `\foreach` loops as well). An example for annotations (with loops) is shown in [Figure 5.6](#). All other graphics that are neither bitmaps nor TikZ graphics are handled using the `svg` approach.

All of this is very convenient indeed, since we can now do everything in  $\text{\LaTeX}$  and `tikz` and basically never have to revisit the base SVG file, unless the graphic itself changes. All the labels stay in the  $\text{\LaTeX}$  source and are therefore also manageable through `git`.

**floatrow** A notable feature is the capability for captions the same width as the float they are attached to. This tends to look much tighter and tidier, see [Figures 5.1](#) and [5.2](#) for a comparison.

Other features are automatic centering of floats, automatic positioning of captions (tables on top, figures below, independent of where the `\caption` command occurs in the source), multiple floats and captions on the side. Depending on the aspect ratio of the image, positioning the caption besides the figure can look (subjectively) pretty, see [Figure 5.4](#).

## Multiple floats

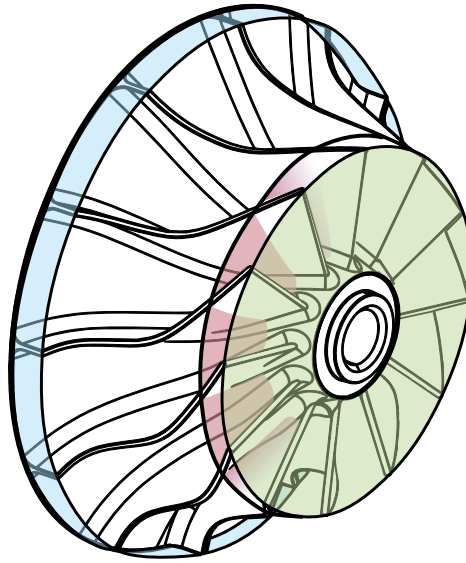
Other possibilities are rather arbitrary arrangements of sub-figures and -captions. For this, see [Figure 5.3](#), which contains the two sub-figures [Figures 5.3a](#) and [5.3b](#). Multiple sub-tables are also possible, see [Table 5.1](#) with its four sub-tables [Tables 5.1a](#) to [5.1d](#).

## Side-Captions

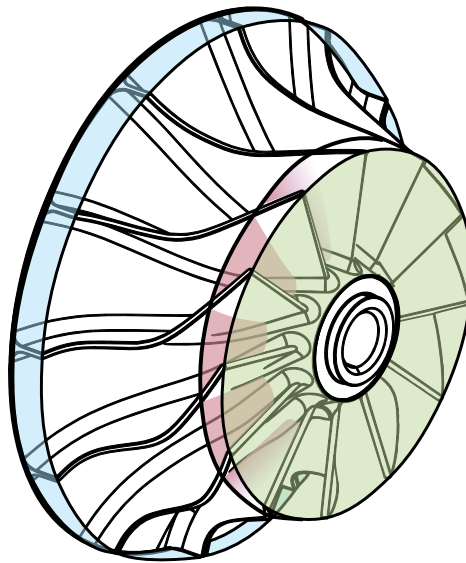
Occasionally, figures and their captions can look disproportionate in combination. In these cases, placing a side-caption might relieve the situation, as shown in [Figure 5.4](#).

## Large Floats

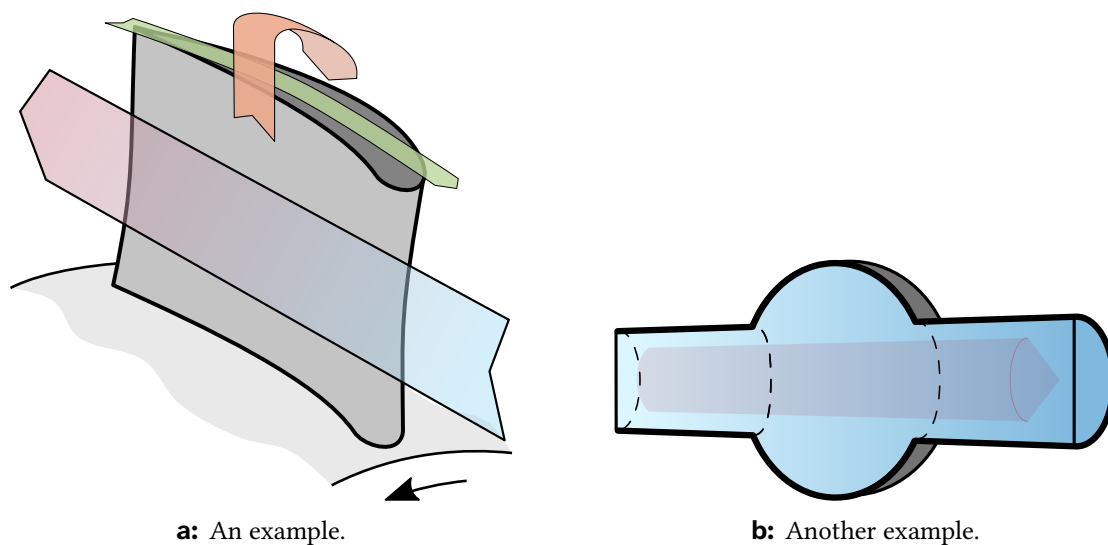
An example for a larger table is shown in [Table 5.2](#). One key aspect there: the S column-type, provided by [siunitx](#). It automatically applies `\num` to each cell, which in turn allows easy printing of things like:  $-3.23 \times 10^{-5}$ . Further, decimal places are accounted for and aligned by automatically. Use `*x{y}` to print column-type *y* (i.e. c) *x* times. No need for tedious repetition.



**Figure 5.1:** Example for a regular caption, spanning the whole width since it is so long. This can quickly look strange, especially if the figure itself is narrow.



**Figure 5.2:** Example for a refined caption, spanning just the width of the float it is attached to, despite the caption being much longer than the float is wide.



**Figure 5.3:** An example for sub-figures.

[.

This field can be used for a reference to a source: Adapted from wherever.]This field can be used for a reference to a source: Adapted from wherever.

**Table 5.1:** Example for multiple tables in one float.

Column One	Column Two	Column One	Column Two
Just normal	a table	Just normal	a table
Nothing	special.	Nothing	special.
a: Table One.		b: Table Two.	
Column One	Column Two	Column One	Column Two
Just normal	a table	Just normal	a table
Nothing	special.	Nothing	special.
c: Table Three.		d: Table Four.	





**Figure 5.4:** A side caption,  
which may also span multi-  
ple lines like demonstrated  
in this rather long caption  
right here.

[.  
CTAN lion drawing by  
Duane Bibby; thanks to  
[ctan.org](http://ctan.org)]CTAN lion drawing  
by Duane Bibby; thanks to  
[ctan.org](http://ctan.org)

**Table 5.2:** Compositions and properties of fuels, as used in Equation 5.1.

Name	Mass fraction $\gamma$							$\rho$	$H_i$
	[-]							[kg/m <sup>3</sup> ]	[MJ/kg]
	C	H	S	O	N	H <sub>2</sub> O	Ash		
Diesel <sup>1</sup>	0.8600	0.1320	0.0060	0.0020 <sup>2</sup>		n.a.	n.a.	840	42.7
Oil EL <sup>1</sup>	0.8570	0.1310	0.0100	0.0020 <sup>2</sup>		n.a.	n.a.	840	42.7
Oil H <sup>1</sup>	0.8490	0.1060	0.0350	0.0100 <sup>2</sup>		n.a.	n.a.	980	40.0
Marine Diesel Oil (MDO) <sup>3</sup>	n.a.	n.a.	0.0150	n.a.	n.a.	0.0030 <sup>4</sup>	0.0001	900	n.a.
Heavy Fuel Oil (HFO) <sup>5</sup>	n.a.	n.a.	0.0350 <sup>6</sup>	n.a.	n.a.	0.0050 <sup>4</sup>	0.0015	1010	n.a.
Light <sup>7</sup>	0.8600	0.1320	0.0060	0.0010	0.0010	0	0	840	Eq. 4.15
Medium <sup>7</sup>	0.8530	0.1269	0.0150	0.0010	0.0010	0.0030	0.0001	900	Eq. 4.15
Heavy <sup>7</sup>	0.8460	0.1025	0.0350	0.0050	0.0050	0.0050	0.0015	1010	Eq. 4.15

**Very long tables** See Table A.1 in Section A.2 for an example of a very large table (which does not float).

## Table Style

In general, use the least ink possible to get your point across. Any more is only noise. This is especially true for tables. For this, compare Table 4.1a to its not-so-blessed twin, Table 5.3:

- do not use vertical lines in tables,
- avoid double lines,
- if in doubt, left-align. If there is no actual reason to center or right-align, refrain from it. Of course, if your language flows right-to-left, like Arabic or Hebrew, this advice is reversed.

For more info, see the package `booktabs`.

## Caption Positioning

Note that table captions (see for example Table 5.1) occur *above* the table no matter the `\caption` command's position in the source code. Per convention, figure captions should appear below, table captions above their bodies. This is handled automatically by `floatrow`. Also note that there is neither a period nor *any* character (no space, no empty line) behind the last caption line in the source code, since those periods are managed globally by the `caption` package. They can therefore be toggled easily.

**Float Footer** There is also a `\floatfoot` command for all floats. This is used to place additional info underneath the caption, for example for references, cf. Figure 5.3.

**Table 5.3:** Dreadful version of Table 4.1a.

<i>Feature</i>	<i>Sample Text</i>
Regular	The quick brown Fox jumps over the lazy Dog 13 times!
<b>Bold</b>	<b>The quick brown Fox jumps over the lazy Dog 13 times!</b>
<i>Italics</i>	<i>The quick brown Fox jumps over the lazy Dog 13 times!</i>
<b><i>Bold Italics</i></b>	<b><i>The quick brown Fox jumps over the lazy Dog 13 times!</i></b>
SMALL CAPITALS	THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG 13 TIMES!
<b>BOLD SC</b>	<b>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG 13 TIMES!</b>
<i>ITALICS SC</i>	<i>THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG 13 TIMES!</i>

## 5.1. TikZ and pgfplots

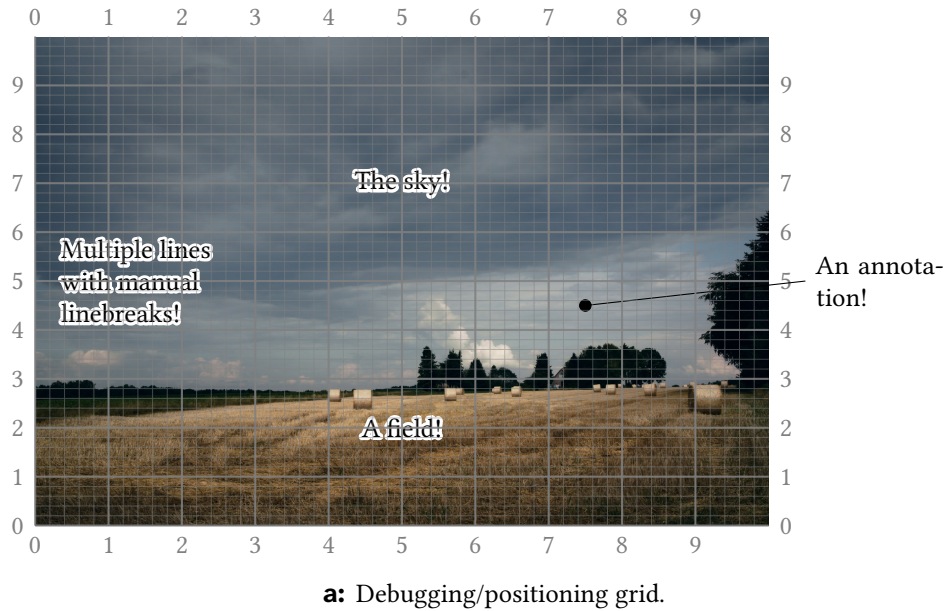
Packages `tikz` and `pgfplots` offer a vast array of features. A select few are presented here.

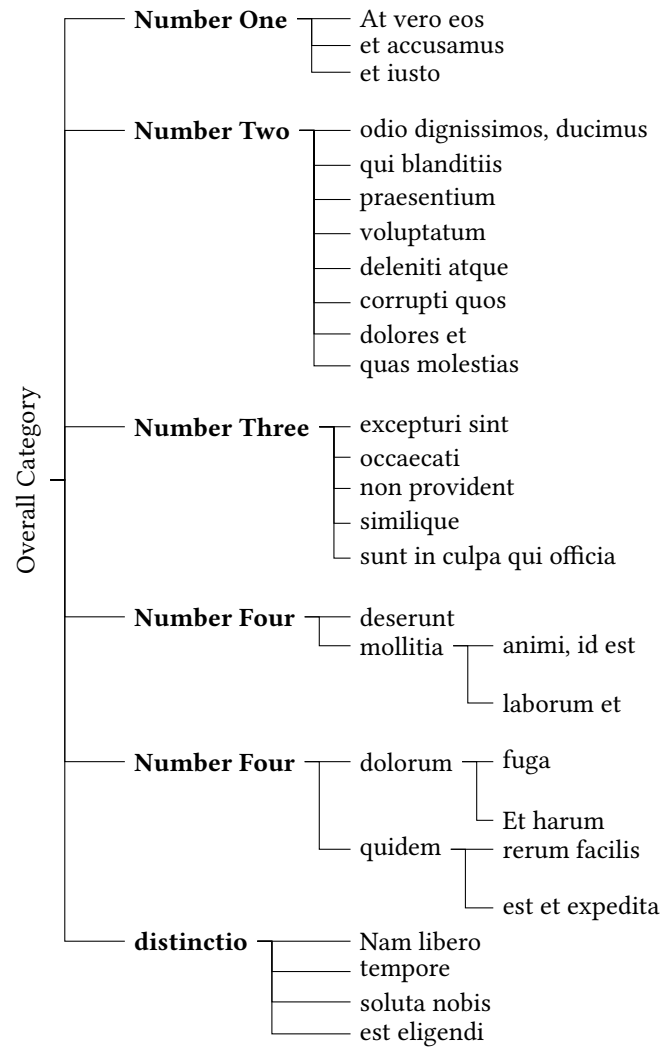
### 5.1.1. Drawing over Bitmaps

When having to rely on bitmaps, one might still want to add additional info. This can be done directly in  $\text{\LaTeX}$ , profiting from all the usual features. In the example here, this is of course the retaining of the text font, but also employing the useful `contour` package to draw legible black-on-white (or vice-versa) text. An example is shown in Figure 5.6. There is a debugging grid functionality to allow for easier positioning of the labels on the graphic. Figure 5.6 also showcases a vertical alignment of sub-figures.

### 5.1.2. Trees

Drawing trees is possible with `forest`, see Figure 5.7. It is a very easy package to get started with. Take a look at the source code. It is deceptively simple, but also highly customizable. Note that the package is stand-alone, but uses TikZ under the hood (like a lot of packages do).

**b:** Result.**Figure 5.6:** Drawing over a bitmap graphic using TikZ in a vertical sub-figure environment.



**Figure 5.7:** Example for a tree.

### 5.1.3. Flow charts

Drawing flow charts is possible with TikZ as well. An example can be found in [Figure 5.8](#). The style definitions for the nodes can be found in the class file.

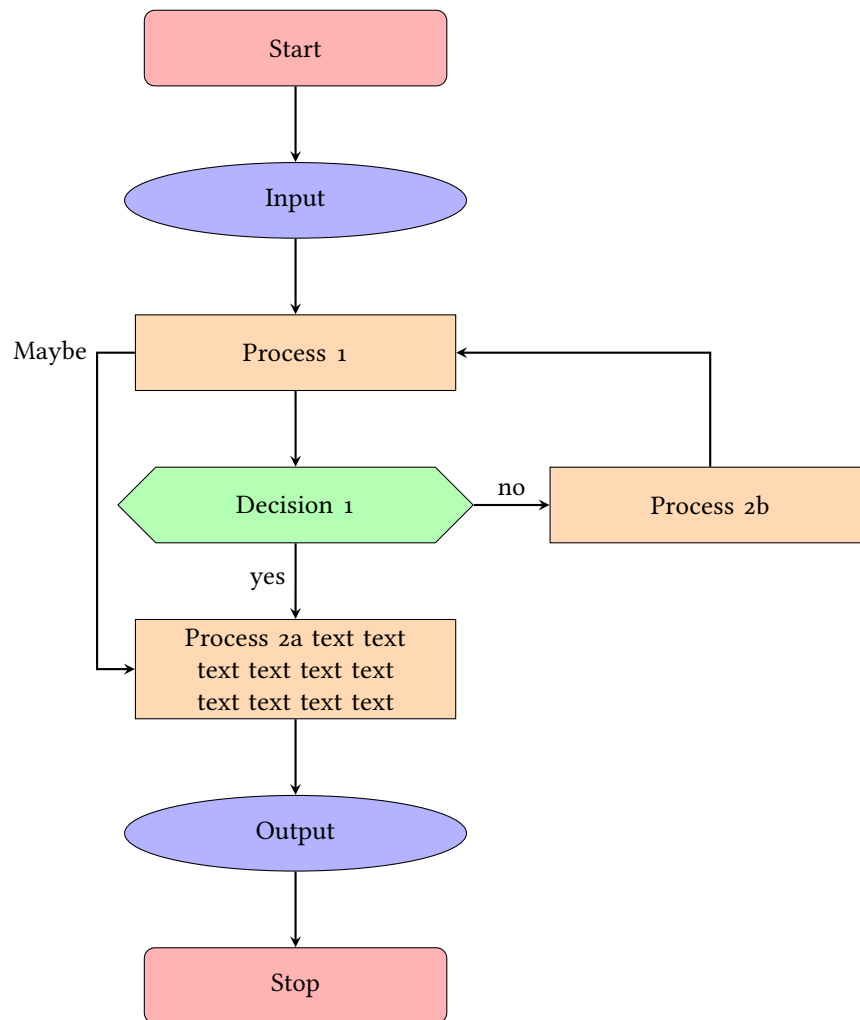
### 5.1.4. Plotting

If you rely on tools like `matlab2tikz`, this is for you. Instead of these outside tools, one can plot *directly* in  $\text{\LaTeX}$ , either through importing external `*.csv` data (for example from experiments) or through computing the plots using `pgfplots` directly from within  $\text{\LaTeX}$ .

#### Directly in $\text{\LaTeX}$

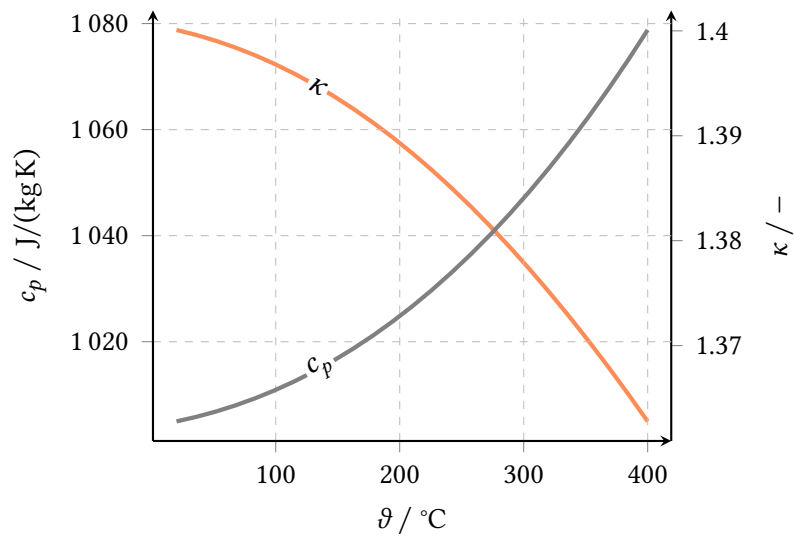
Plotting and computing the return values directly in  $\text{\LaTeX}$  is achieved through the `declare function` command. While the functionality is limited (owed to the limitations of  $\text{\TeX}$ , which is of course not meant for this sort of thing), it may still save a lot of time and headaches. Finding and specifying the correct settings for `pgfplots` can take a lot of time. This is already taken care of for this document. As such, a plot from existing `*.csv` data can be set up in a handful of lines using one of the built-in styles, like `regularplot`. Plotting from a TikZ function is demonstrated in [Figures 5.9](#) and [5.10](#).

**Contour Plots** Contour plots are too much to handle for `pgfplots` itself. This is owed to the limitations of the underlying  $\text{\TeX}$  engine. What the math and computation engine of `pgfplots` does is amazing, but in the end,  $\text{\TeX}$  is a typesetting system, not a general-purpose programming language. However, `pgfplots` has an option to delegate computations to the external `gnuplot` tool. In analogy to `svg`, using it requires `gnuplot` itself to be installed and on the `$PATH`. [??](#) shows an example for a plot entirely specified in the  $\text{\LaTeX}$  source. All the functions required are defined using `pgfplots` functionalities! “Only” the computation itself is outsourced to `gnuplot`. The resulting plot is two-dimensional, but the underlying logic is three-dimensional, where one dimension was reduced, forming contours. This is where the complexity resides. The entire plot is just 200 lines of code, with really only half that being lines that effectively do something useful.

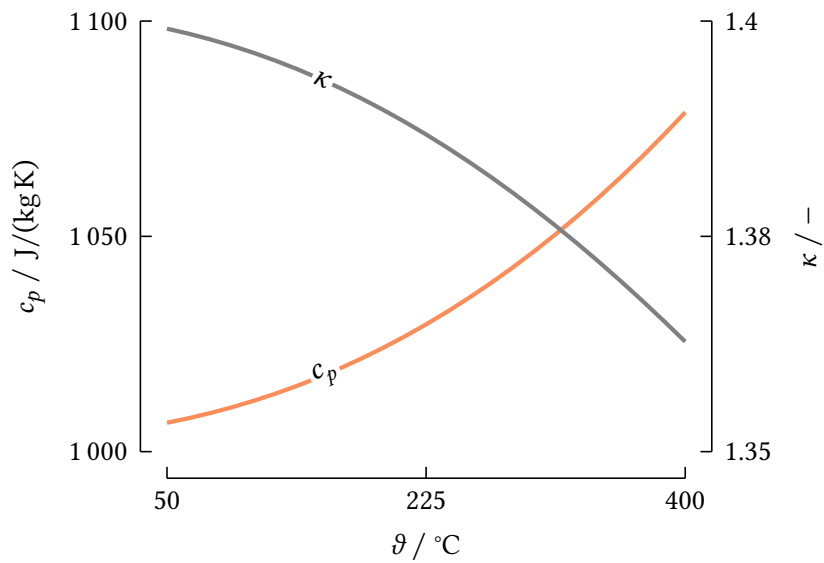


**Figure 5.8:** Example for a Flowchart.

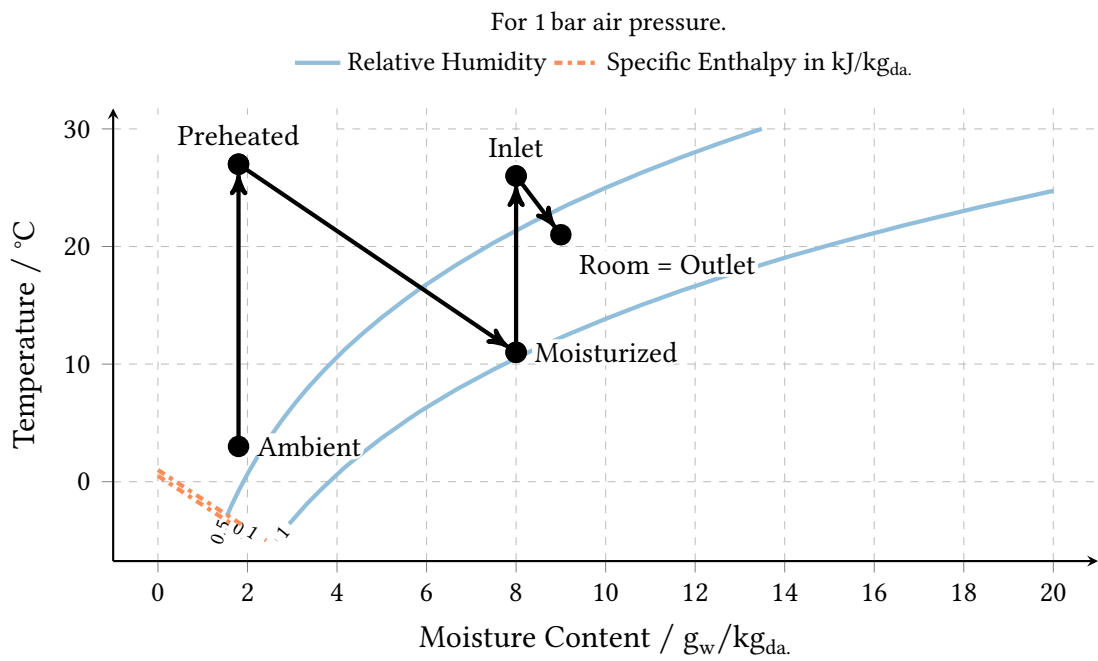




**Figure 5.9:** Caloric parameters of air. **Avoid legends** and put info where it belongs, improving legibility (less back-and-forth for the eye). This plot is using the custom `regularplot` style.



**Figure 5.10:** Same content as [Figure 5.9](#) in “Tufte-like” for a modern, minimalist look where precision counts less than the overall message. For more info, refer to its namesake, TUFTE.



**Figure 5.11:** Wastegate implementation in a feedback-loop in MATLAB/Simulink as an example for a TikZ diagram.

**Time-series** Time-series plots are straightforward to implement. **pgfplots**, using its `dateplot` library, can automatically parse and plot dates. For this to work best and most reliably, dates and times should be in ISO 8601 format:

```
YYYY-MM-DDThh:mm:ss :
2020-05-19T12:15:31Z,
```

where Z is time-zone information and T is an arbitrary, but fixed separator. It can also be a simple space. Use just the date *or* time part when appropriate.

This format is unambiguous and understood worldwide as well as, most importantly here, by computers. No extra string and date parsing is required, it will just work in a lot of cases, not only for **pgfplots** like here. Further, it is often the standard output format (or close to it) of measurement equipment anyway, so there is not even a need to modify it. An example is shown in [Figure 5.12](#).

## Importing CSV

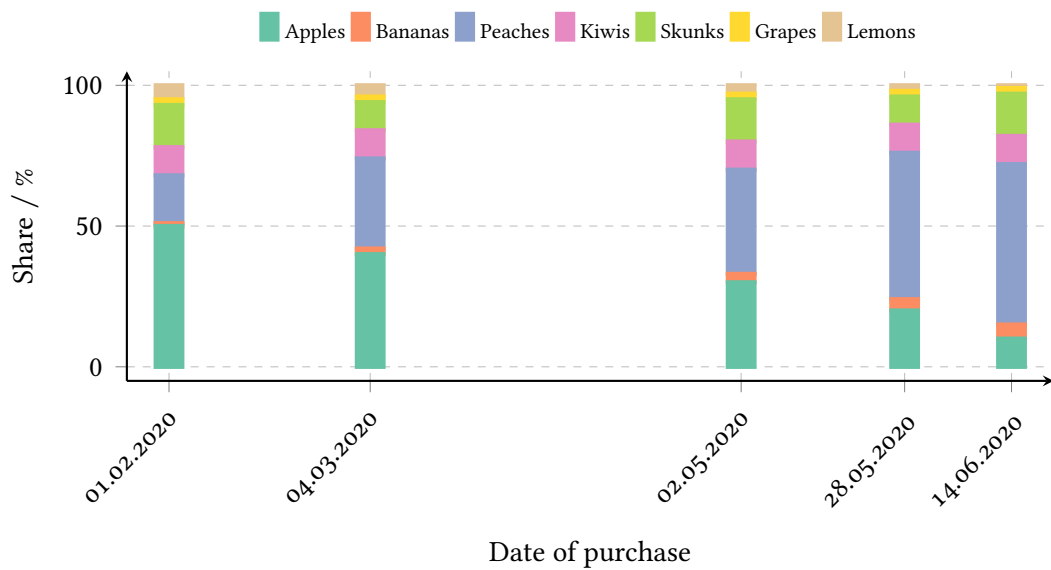
Often, one wants to plot data from files. The better behaved the file is (meaningful headers, no junk rows), the easier that is. In [Figure 5.13](#), `y=<column header>` is all that has to be specified for the column corresponding to that name to be automatically chosen, with no confusion about indices or column numbers.

## Using MATLAB2TikZ

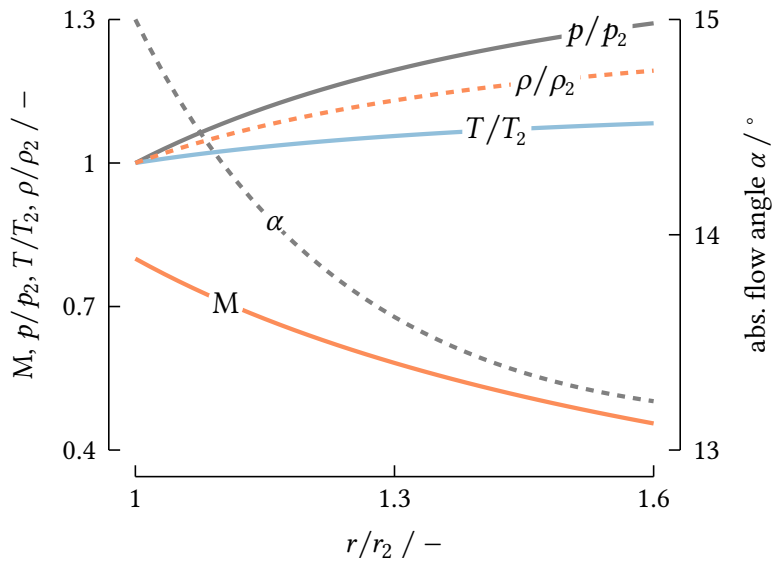
**MATLAB2TikZ** is a tool to convert MATLAB figures to  $\text{\LaTeX}$ , see [Figure 5.14a](#). Notice that by default, it exports using whitespace (tabs) as column separators, which might differ from what is set as a global option in `\pgfplotstableset`.

[Figure 5.15a](#) is the same plot, but generated directly in  $\text{\LaTeX}$ , without relying on **MATLAB2TikZ**—using the latter, a plethora of things break since it hard-codes a bunch of stuff.

[Figure 5.15a](#) is only around 30 lines of code (arrived at after only a couple of hours...)!



**Figure 5.12:** Example automatic timeseries plot. Note the automatic spacing-out according to the actual time deltas, and the automatic conversion of timestamps to human-friendly versions, to whatever specification the author chooses. Also note the colors: here, *distinction* is important and a *qualitative* palette is chosen. A *sequential* or a *diverging* palette would have been less suited (some would say plain wrong).



**Figure 5.13:** Plotting from CSV data for a diffuser.

### 5.1.5. TikZ and Text

TikZ content can also be intertwined with text using `\tikzmark`. This is illustrated in Equation 5.1. Note that this procedure needs two compilation runs, since the label positions need to be written to an auxiliary file first.

$$\gamma_C + \gamma_H + \gamma_S + \gamma_O + \gamma_N + \gamma_{H_2O} + \gamma_{ash} := 1. \quad [5.1]$$

$\gamma_C$   
Carbon

$\gamma_H$   
Hydrogen

$\gamma_S$   
Sulphur

$\gamma_O$   
Oxygen

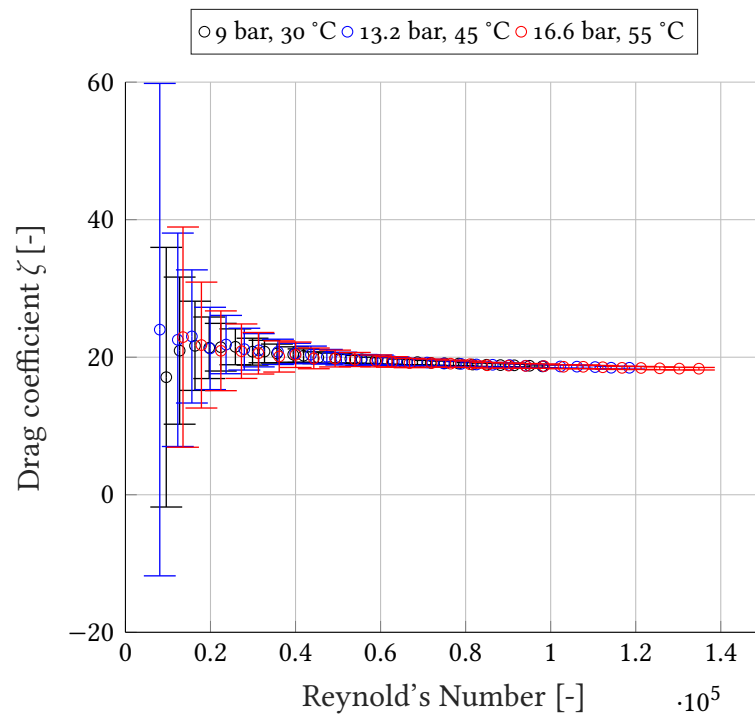
$\gamma_N$   
Nitrogen

$\gamma_{H_2O}$   
Water

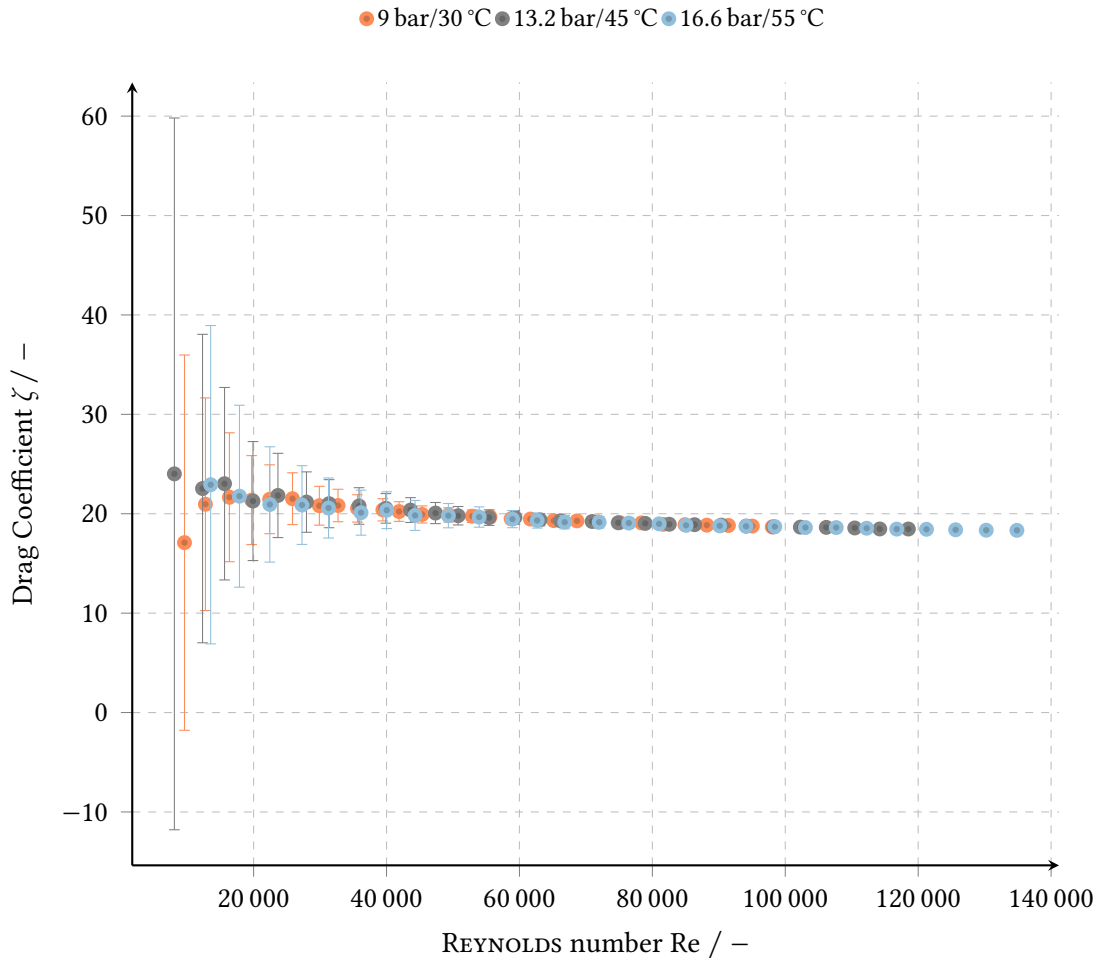
$\gamma_{ash}$   
Ash

### 5.1.6. Regular TikZ pictures

TikZ really is not meant for arbitrary graphics. The more free-form images shown in this document were created using InkScape. Still, “drawing” in TikZ is much preferred and

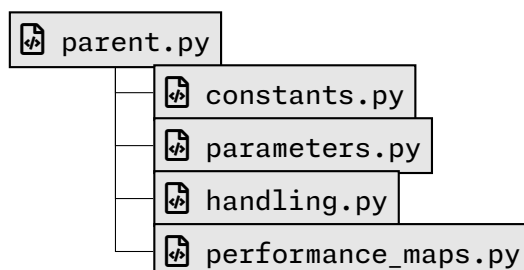


**Figure 5.14a:** A vanilla `MATLAB2TikZ` example, imported here without changes except those to allow successful compilation (see commit 56556a0: set `table/col sep=space`). While it works, the style created in MATLAB conflicts with the local one, and you miss out on many useful features, like `siunitx` or `glossaries-extra`. A more frictionless approach is to export plain-text (CSV) data from MATLAB, and import it into  $\text{\LaTeX}$ , see Figures 5.13 and 5.15a.



**Figure 5.14b:** The same data as Figure 5.14a, but plotted directly in  $\text{\LaTeX}$  using `pgfplots` for full styling control.

easier when the images are somewhat programmatic, aka there are straight corners, edges and turns, equal distances, and everything is a bit “block-like”, repetitive. For example, a small file structure diagram:



Note how `tikzpicture` environments do not have to be contained in floats.

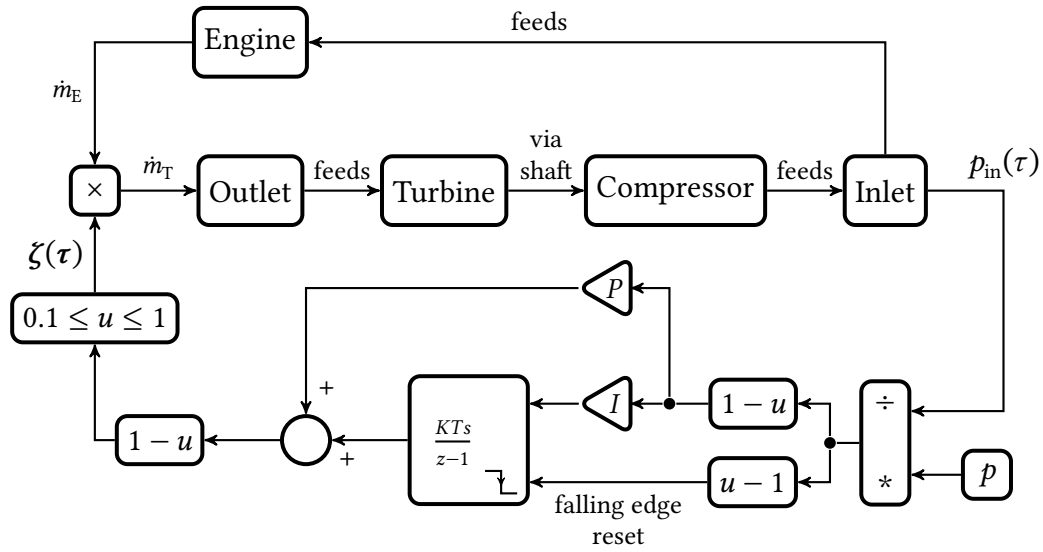
More TikZ examples are shown in [Figures 5.11](#), [5.16](#) and [5.17](#).

**Included shapes** This repository includes custom-made shapes for thermodynamic applications. These can be used like many other TikZ elements, for example by positioning them somewhere on the canvas, connecting them to other elements, rotating them *etc.* In that sense, they work like usual TikZ elements (just buggier...). There are a couple of advantages:

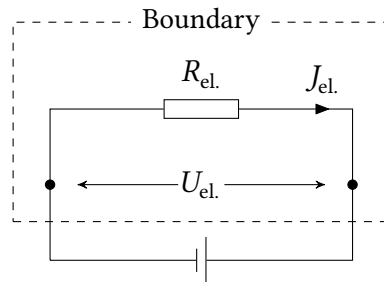
1. unified looks: no more drawing these in Inkscape, where they come out slightly dissimilar every time,
2. tight integration with TikZ, allowing to use all its other features,
3. very fast generation of drawings once some familiarity is gained; with Inkscape or other outside tools, one can also become fast, but it is hard to beat a  $\text{\LaTeX}$ -internal approach.

Examples are shown in [Figures 5.18a](#) and [5.18b](#). Refer to their source code to see how more or less easily they are created.

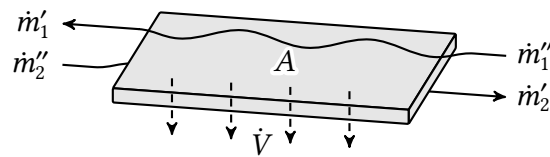




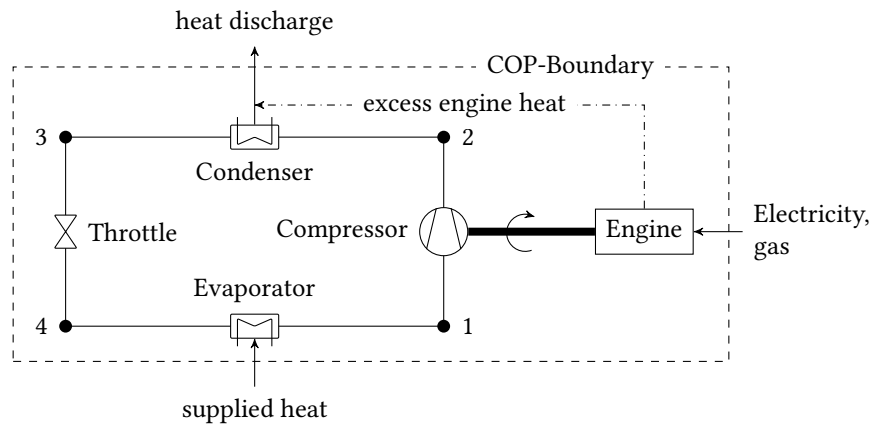
**Figure 5.15a:** Same data as Figure 5.14a but plotted in  $\text{\LaTeX}$  directly, using only the provided raw data. Note how this allows for a consistent style and unlocks all other, usual features found in this template. Also take a look at the source data to see what *tidy data* looks like, a data format most suitable for data processing.



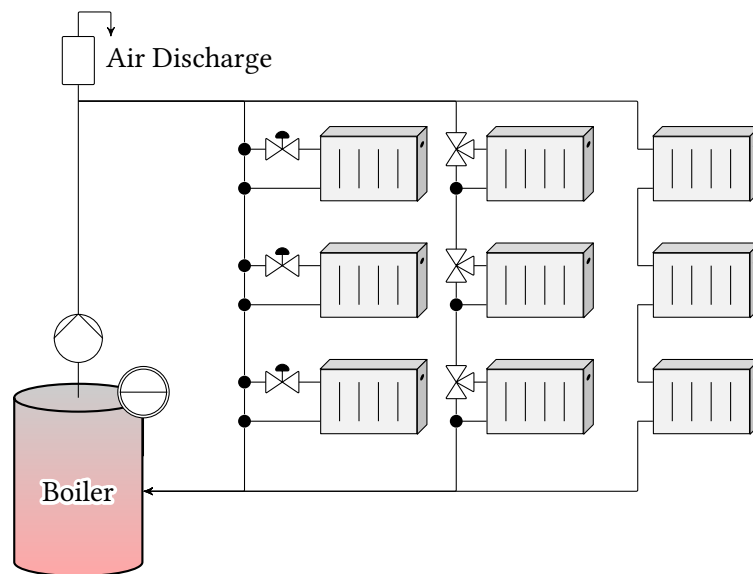
**Figure 5.16:** Example for the `circuits.ee.IEC` TikZ library.



**Figure 5.17:** Example for a three-dimensional TikZ drawing using the `3d` library.



**Figure 5.18a:** Example for a thermodynamic device drawing using TikZ. It relies heavily on the custom-made library of shapes.



**Figure 5.18b:** Example TikZ shapes.

### 5.1.7. Inkscape

Having seen what `TikZ` is good at, [Figures 5.2 to 5.4](#) are good examples for when Inkscape might be the better choice: three-dimensional, curvy drawings.

[Figure 5.2](#) is a vectorized bitmap. Inkscape can detect edges and contrasts in bitmaps and replicate those lines in vector format (Path > Trace Bitmap).

## 5.2. Example Boxes

As a special gimmick, there is an environment for examples (can also be renamed). It may be useless now, but you can alter it to suit your needs; the skeleton is there for you. It even has its own list, like the list of figures. For an example box, see [Example 5.2.1](#).

### Example 5.2.1: I am a useless box

There can be pretty much any content in here. Math works — as we can see,

$$1 = 1 \qquad [5.2]$$

still holds true after all these years!

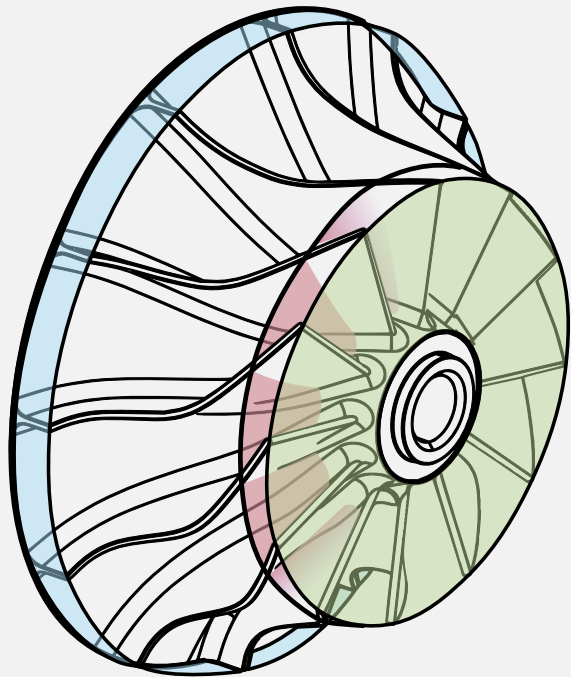
If the `float` setting is set, this box also floats. Inserting other floats in here will then cause `TEX` to have a massive fit (floats inside floats do not make sense). This is circumvented setting the `[H]` flag, saying “this float is not really a float, pin it down *right* here”.

**In any other context, setting any such flag is code smell/poor style.**

These are often used wrong and just as often set prematurely and then reset countless times, all the while `TEX` complains (rightfully so) about the poor spacing introduced by forcing float positions, instead of letting `TEX` take care of it. For the love of God, let `TEX` do its job in placing the floats. Truth be told, they will on occasion not be placed where you need or want them. Keep working. At the very end, when all is done, go ahead and change the few *truly*

misplaced floats manually, by shoving them about in the source code (still not using `htb!` flags). This ensures minimal pain and maximum usage of  $\text{\LaTeX}$ 's spacing and placing capabilities.

Anyway — here is such a float within an example box:



**Figure 5.19:** Side-captions are still possible. So are labels.

Note how using `\linewidth` as a length, not the global constant `\textwidth`, figures can be scaled according to the current context.

This box even breaks across pages if so required. Should this turn out ugly, some manual action is certainly required.

## 6. Code Syntax Highlighting

To properly typeset code in  $\text{\LaTeX}$ , we use the `minted` package. It relies on Python, and calls in outside help for syntax highlighting using Python's `pygments` package. As such, it requires `--shell-escape` to compile, and of course Python. The latter can be a pain in the buttocks to set up; Docker usage is especially useful here. Since `pygments` has nothing to do with  $\text{\LaTeX}$  (whose ecosystem is a sad mess), but is instead a regular old Python package, chances are the language of your choice is not only available but also well-supported!<sup>1</sup>

**Color Scheme** Note how the color scheme is the same throughout languages. The idea is that keywords of similar importance, status or semantics are highlighted uniformly. For example, keywords for class and function definitions, like `class` for Python or `classdef` for MATLAB.<sup>2</sup> Another example are error-handling and -throwing keywords, which many languages offer. All these different types should be identified and treated equally. `minted` is a widely used package and knows about a lot of languages. You can check it out at

<https://pygments.org/demo/>.

If the current colors are not to your liking they can be changed easily using `minted`'s `style` option. Refer to the comments in the source code on how to see available styles.

---

<sup>1</sup>For example, `listings`, the inferior alternative to `minted`, still had no Python 3 support (only basic 2) in 2020, at the time of originally writing this. At that point, Python 3 was over 10 years old already and Python 2 was end-of-life.

<sup>2</sup>Notice how these keywords were created using an *inline* listing, like: `y = [file_patterns[x] for x in ["send", "help"]]`.

**References** Individual code lines can also be referenced. For example, we find a `return` statement on line 24 in Code Listing 6.1. That line is also highlighted, using `minted`'s `highlightlines` option.

## 6.1. Python

The following examples are not always complete or functioning, they are only supposed to showcase the available syntax highlighting. The base style looks like:

```

1  def get_nonempty_line(
    lines: Iterable[str],
    last: bool = True
) -> str:
5     if last:
        lines = reversed(lines)
    return next(line for line in lines if line.rstrip())

```

It is intended for (small) samples of code that flow into the surrounding text. A second feature are code listings as regular floats, as demonstrated in Code Listing 6.1. As floats, they behave like any other figure, table *etc.*

Code Listing 6.2 showcases breaking across pages, imported from a file, probably best suited for an appendix:

**Code Listing 6.2:** This is a page breaking code, with a caption, which is imported from a file!.

```

1  def ansi_escaped_string( A random reference: Figure 4.1
    string: str,
    *,
    effects: Union[List[str], None] = None,
5     foreground_color: Union[str, None] = None,
    background_color: Union[str, None] = None,
    bright_fg: bool = False,
    bright_bg: bool = False,
) -> str:
10     """Provides a human-readable interface to escape strings for terminal
        ↪ output.

```

**Code Listing 6.1:** This is a caption. Listings cannot be overly long since floats do not page-break. Therefore is the longlisting environment..

```

1  import json
   import logging.config
   from pathlib import Path

5  from resources.helpers import path_relative_to_caller_file


$$\text{\LaTeX can go in here: } \sum_{i=1}^n a + \frac{\pi}{2}$$


10 def set_up_logging(logger_name: str) -> logging.Logger:
    """Set up a logging configuration."""
    config_filepath = path_relative_to_caller_file("logger.json") # same directory

    try:
        with open(config_filepath) as config_file:
            config: dict = json.load(config_file)
            logging.config.dictConfig(config)
    except FileNotFoundError:
        logging.basicConfig(
            level=logging.INFO, format="[%(asctime)s: %(levelname)s] %(message)s"
20         )
        logging.warning(f"Using fallback: no logging config found at
            ↳ {config_filepath}")
        logger_name = __name__

    return logging.getLogger(logger_name)

```

Using ANSI escape characters, the appearance of terminal output can be  
↳ changed. The  
escape chracters are numerical and impossible to remember. Also, they  
↳ require  
special starting and ending sequences. This function makes accessing that  
↳ easier.

#### Args:

*string*: The input string to be escaped and altered.  
*effects*: The different effects to apply, e.g. underlined.  
*foreground\_color*: The foreground, that is text color.  
*background\_color*: The background color (appears as a colored block).  
*bright\_fg*: Toggle whatever color was given for the foreground to be  
↳ bright.  
*bright\_bg*: Toggle whatever color was given for the background to be  
↳ bright.

#### Returns:

A string with requested ANSI escape characters inserted around the  
↳ input string.

"""

```
def pad_sgr_sequence(sgr_sequence: str = "") -> str:
```

"""Pads an SGR sequence with starting and end parts.

To 'Select Graphic Rendition' (SGR) to set the appearance of the  
↳ following

terminal output, the CSI is called as:

CSI *n m*

So, '*m*' is the character ending the sequence. '*n*' is a string of

↳ parameters, see

dict below.

#### Args:

*sgr\_sequence*: Sequence of SGR codes to be padded.



*Returns:*

*Padded SGR sequence.*

"""

```
control_sequence_introducer = "\x1B[" # hexadecimal '1B'
select_graphic_rendition_end = "m" # Ending character for SGR
return control_sequence_introducer + sgr_sequence +
    ↪ select_graphic_rendition_end
```

*# Implement more as required, see*

*# [https://en.wikipedia.org/wiki/ANSI\\_escape\\_code#SGR\\_parameters](https://en.wikipedia.org/wiki/ANSI_escape_code#SGR_parameters).*

```
sgr_parameters = {
    "underlined": 4,
}
```

*sgr\_foregrounds = { # Base hardcoded mapping, all others can be derived*

```
"black": 30,
"red": 31,
"green": 32,
"yellow": 33,
"blue": 34, 30 + 4
"magenta": 35,
"cyan": 36,
"white": 37,
```

```
}
```

*# These offsets convert foreground colors to background or bright color*

*↪ codes, see*

*# [https://en.wikipedia.org/wiki/ANSI\\_escape\\_code#Colors](https://en.wikipedia.org/wiki/ANSI_escape_code#Colors)*

```
bright_offset = 60
```

```
background_offset = 10
```

```
if bright_fg:
```

```
    sgr_foregrounds = {
        color: value + bright_offset for color, value in
        ↪ sgr_foregrounds.items()
    }
```

```

75     if bright_bg:
        background_offset += bright_offset

sgr_backgrounds = {
    color: value + background_offset for color, value in
        ↪ sgr_foregrounds.items()
}

80 # Chain various parameters, e.g. 'ESC[30;47m' to get white on black, if 30
    ↪ and 47
# were requested. Note, no ending semicolon. Collect codes in a list first.
sgr_sequence_elements: List[int] = []

85 if effects is not None:
    for sgr_effect in effects:
        try:
            sgr_sequence_elements.append(sgr_parameters[sgr_effect])
        except KeyError:
            raise NotImplementedError(
90                 f"Requested effect '{sgr_effect}' not available."
            )
if foreground_color is not None:
    try:
        sgr_sequence_elements.append(sgr_foregrounds[foreground_color])
95     except KeyError:
        raise NotImplementedError(
            f"Requested foreground color '{foreground_color}' not
            ↪ available."
        )
if background_color is not None:
100    try:
        sgr_sequence_elements.append(sgr_backgrounds[background_color])
    except KeyError:
        raise NotImplementedError(

```

```

105         f"Requested background color '{background_color}' not
            ↪ available."
        )

        # To .join() list, all elements need to be strings
        sgr_sequence: str = "".join(str(sgr_code) for sgr_code in
            ↪ sgr_sequence_elements)

110 reset_all_sgr = pad_sgr_sequence() # Without parameters: reset all
            ↪ attributes
        sgr_start = pad_sgr_sequence(sgr_sequence)
        return sgr_start + string + reset_all_sgr

```

## 6.2. MATLAB

This section contains example code for MATLAB, for example:

```

1  %{
    Universal Gas Constant for SIMULINK.
%}
R_m = Simulink.Parameter;
5  R_m.Value = 8.3144598;
    R_m.Description = 'universal gas constant';
    R_m.DocUnits = 'J/(mol*K)';

```

Of course, floats (see [Code Listing 6.3](#)) are available as well. So are longer sections, like [Code Listing 6.4](#).

**Code Listing 6.4:** This is a page breaking matlab code, with a caption, which is imported from a file!.

```

1  fullfilepath = mfilename('fullpath');
    [filepath, filename, ~] = fileparts(fullfilepath);

    %% Begin Dialogue
5  %{

```

**Code Listing 6.3:** A class definition in MATLAB, from [MATHWORKS 2020].

```

1  classdef BasicClass
    properties
        Value {mustBeNumeric}
    end
    methods
        function r = roundOff(obj)
            r = round([obj.Value],2);
        end
        function r = multiplyBy(obj,n)
            r = [obj.Value] * n;
        end
    end
end
end

```

```

Extract Data from user-specified input. Can be either a File that is run
(an *.m-file), variables in the Base Workspace or existing data from a
previous run (a *.MAT-file). All variables are expected to be in Table
format, which is the data type best suited for this work. Therefore, we
force it. No funny business with dynamically named variables or naked
matrices allowed.
%}
pass_data = questdlg({'Would you like to pass existing machine data?', ...
    'Its data would be used to compute your request.', ...
    'Regardless, note that data is expected to be in ',...
    'Tables named '', dflt.comp, '' and '', dflt.turb, ''.'}, '',...
    'If you choose no, existing values are used.}', ...
    'Machine Data Prompt', btnFF, btnWS, btnNo, btnFF);

switch pass_data
case btnFF
    prompt = {'File name containing the Tables:', ...
        'Compressor Table name in that file:',...
        'Turbine Table name in that file:'};
    title = 'Machine Data Input';
    dims = [1 50];

```

```

definput = {dflt.filename.data, dflt.comp, dflt.turb};
machine_data_file = inputdlg(prompt, title, dims, definput);
if isempty(machine_data_file)
    fprintf('%s] You left the dialogue and function.\n',...
        datestr(now));
    return
end
run(machine_data_file{1});%spills file contents into funcWS
case btnWS
    prompt = {'Base Workspace Compressor Table name:',...
        'Base Workspace Turbine Table name:'};
    title = 'Machine Data Input';
    dims = [1 50];
    definput = {dflt.comp, dflt.turb};
    machine_data_ws = inputdlg(prompt, title, dims, definput);
    if isempty(machine_data_ws)
        fprintf('%s] You left the dialogue and function.\n',...
            datestr(now));
        return
    end
case btnNo
    boxNo = msgbox(['Looking for stats in ''', dflt.filename.stats, ...
        '''.'], 'Using Existing Data', 'help');
    waitfor(boxNo);
    try
        stats = load(dflt.filename.stats);
        stats = stats.stats;
    catch ME
        switch ME.identifier
            case 'MATLAB:load:couldNotReadFile'
                warning(['File ''', dflt.filename.stats, '' not ',...
                    'found in search path. Make sure it has been ',...
                    'generated in a previous run or created ',...
                    'manually. Resorting to hard-coded data ',...
                    'for now.']);
                a_b = 200.89;

```

```

65         x_c = 0.0012;
           f_g = 10.0;
           otherwise
             rethrow(ME);
         end
       end
     case ''
70       fprintf('%s] You left the dialogue and function.\n',datestr(now));
       return
       otherwise%Only gets here if buttons are misconfigured
         error('This option is not coded, should not get here.');
```

end

### 6.2.1. MATLAB/Simulink icons

For an older project, MATLAB/Simulink vector icons were created. They are included here at the off-chance that someone else might find a use for these.

- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 

- 
- 
- 
- 

## 6.3. Modelica

Naturally, floating and all other environments and styles are also available for Modelica. The syntax highlighting for a few basic code samples<sup>3</sup> looks like:

```

1  x := 2 + y;
   x + y = 3 * z;

model FirstOrder
5    parameter Real c=1 "Time constant";
    Real x (start=10) "An unknown";
equation
    der(x) = -c*x "A first order differential equation";
end FirstOrder;

10
type Voltage = Real(quantity="ElectricalPotential", unit="V");
type Current = Real(quantity="ElectricalCurrent", unit="A");

connector Pin "Electrical pin"
15    Voltage v "Potential at the pin";
    flow Current i "Current flowing into the component";
end Pin;

model Capacitor
20    parameter Capacitance C;
    Voltage u "Voltage drop between pin_p and pin_n";
    Pin pin_p, pin_n;

```

---

<sup>3</sup>WIKIPEDIA CONTRIBUTORS 2021.

```

equation
  0 = pin_p.i + pin_n.i;
  u = pin_p.v - pin_n.v;
  C * der(u) = pin_p.i;
end Capacitor;

model SignalVoltage
  "Generic voltage source using the input signal as source voltage"
  Interfaces.PositivePin p;
  Interfaces.NegativePin n;
  Modelica.Blocks.Interfaces.RealInput v(unit="V")
    "Voltage between pin p and n (= p.v - n.v) as input signal";
  SI.Current i "Current flowing from pin p to pin n";
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end SignalVoltage;

model Circuit
  Capacitor C1(C=1e-4) "A Capacitor instance from the model above";
  Capacitor C2(C=1e-5) "A Capacitor instance from the model above";
  ...
equation
  connect(C1.pin_p, C2.pin_n);
  ...
end Circuit;

```

## 6.4. Lua

Files can also be read into L<sup>A</sup>T<sub>E</sub>X directly. For example, the following is some current Lua code *used for this very document*:

```

1  --[[
    The following import is not required since in LuaTeX's '\directlua'
    ↪ environment,

```



*`token` etc. is already available. However, this is nice to keep linters from complaining about an undefined variable.*

```
--]]
```

```
local token = require("token")
local texio = require("texio")
local status = require("status")
```

```
--[[
```

*Trying to incorporate dynamic values into certain newcommand macros. Their contents are set at build-time according to environment variables. This is*  
*↪ useful*

*for automatic workflows in CI environments. See also:*  
*<https://tex.stackexchange.com/a/1739/120853>.*

*An alternative to using environment variables are command line arguments:*  
*<https://tex.stackexchange.com/a/18813/120853>*

*However, this seems more error-prone and requires more steps, e.g. piping*  
*↪ arguments*  
*to `lua<sub>l</sub>atex` through `latexmk` first, etc.*

*The previous approach was to `sed` for certain `newcommand` definitions in an additional CI job. This was much more error-prone (bash scripting) and less easily expanded than the below approach.*

*LuaTeX provides excellent access to TeX, making this implementation much*  
*↪ easier.*

```
--]]
```

```
local function get_cmd_stdout(cmd)
  -- See: https://stackoverflow.com/a/326715/11477374
  local fh = assert(io.popen(cmd))
  local first_line = assert(fh:read())
  fh:close()
  return first_line
end
```

```
local function first_env_value(env_spec)
  if not env_spec then
    return nil, nil
  end
```

```

end
if type(env_spec) == "string" then
    env_spec = { env_spec }
end
40 for _, name in ipairs(env_spec) do
    local value = os.getenv(name)
    if value and value ~= "" then
        return value, name
    end
end
45 end
return nil, nil
end

local function normalize_ref(ref)
50 if not ref then
    return ref
end
local cleaned = ref:gsub("^refs/heads/", "")
cleaned = cleaned:gsub("^refs/tags/", "")
55 cleaned = cleaned:gsub("^refs/remotes/", "")
return cleaned
end

local function shorten_sha(sha)
60 if not sha or #sha < 7 then
    return sha
end
return sha:sub(1, 7)
end

65 -- Environment variables as used e.g. in GitLab CI, GitHub Actions, etc.
-- Otherwise, e.g. when developing locally, use commands as a fallback.
local macro_content_sources = {
    GitRefName = {
70     env = {
        "CI_COMMIT_REF_NAME",

```

```

        "GITHUB_REF_NAME",
        "GITHUB_HEAD_REF",
        "GITHUB_REF",
    },
    cmd = "git rev-parse --abbrev-ref HEAD",
    process_env = function(value)
        return normalize_ref(value)
    end,
},
GitShortSHA = {
    env = {
        "CI_COMMIT_SHORT_SHA",
        "GITHUB_SHA",
    },
    cmd = "git rev-parse --short HEAD",
    process_env = function(value)
        return shorten_sha(value)
    end,
    process_cmd = function(value)
        return shorten_sha(value)
    end,
},
GitLongSHA = {
    env = {
        "CI_COMMIT_SHA",
        "GITHUB_SHA",
    },
    cmd = "git rev-parse HEAD",
    allow_empty = true,
},
GitHostPagesURL = {
    env = {
        "GITHUB_PAGES_URL",
        "GITHUB_REPOSITORY",
    },
    cmd = "git remote get-url origin",

```

```

110     process_env = function(value, name)
        if name == "GITHUB_PAGES_URL" then
            return value
        elseif name == "GITHUB_REPOSITORY" then
            local user, repo = value:match("([^\s/]+)/(.+)")
            if user and repo then
                return "https://" .. user .. ".github.io/" .. repo
            end
        end
        return nil
    end,
    process_cmd = function(url)
120         local user, repo = url:match("github%.com[:/](^[^/]+)/([^\s/]+)")
        if user and repo then
            repo = repo:gsub("%.git$", "")
            return "https://" .. user .. ".github.io/" .. repo
        end
125         return nil
    end,
    allow_empty = false,
},
GitRepositorySlug = {
130     env = {
        "GITHUB_REPOSITORY",
    },
    cmd = "git remote get-url origin",
    process_env = function(value, name)
135         return value
    end,
    process_cmd = function(url)
        local user, repo = url:match("github%.com[:/](^[^/]+)/([^\s/]+)")
        if user and repo then
140             repo = repo:gsub("%.git$", "")
            return user .. "/" .. repo
        end
        return nil
    end

```

```

    end,
    allow_empty = false,
},
GitProjectName = {
    env = {
        "GITHUB_REPOSITORY",
    },
    cmd = "git remote get-url origin",
    process_env = function(value, name)
        local user, repo = value:match("([^/]+)/(.+)")
        return repo
    end,
    process_cmd = function(url)
        local user, repo = url:match("github%.com[:/](^[^/]+)/([^/]+)")
        if user and repo then
            repo = repo:gsub("%.git$", "")
            return repo
        end
        return nil
    end,
    allow_empty = false,
},
}

for macro_name, content_sources in pairs(macro_content_sources) do
    local content = nil
    local source = nil

    -- First, try environment variables
    local env_content, env_name = first_env_value(content_sources.env)
    if env_content then
        texio.write_nl("Found environment variable '" .. env_name .. "' for " ..
            ↪ macro_name .. ".")
        content = env_content
        source = "env"
        if content_sources.process_env then

```

```

        content = content_sources.process_env(content)
180     end
    else
        local env_names = content_sources.env
        if type(env_names) == "table" then
            env_names = table.concat(env_names, ", ")
185     end
        texio.write_nl("No environment variable found for " .. macro_name .. "
            ↳ (checked: " .. env_names .. "). Trying Git command fallback.")
    end

    -- If no env, try Git command if shell escape enabled
190 if not content and status.shell_escape == 1 then
        local cmd_success, cmd_stdout = pcall(get_cmd_stdout, content_sources.cmd)
        if cmd_success and cmd_stdout then
            texio.write_nl("Git command '" .. content_sources.cmd .. "' succeeded
                ↳ for " .. macro_name .. ".")
            content = cmd_stdout
195         source = "cmd"
            if content_sources.process_cmd then
                content = content_sources.process_cmd(content)
            end
        else
            texio.write_nl("Git command '" .. content_sources.cmd .. "' failed for
                ↳ " .. macro_name .. ".")
        end
    elseif not content then
        texio.write_nl("Shell escape disabled, cannot use Git command fallback for
            ↳ " .. macro_name .. ".")
    end

205
    -- If still no content, log failure and set to empty (unless allow_empty is
    ↳ false)
    if not content then
        if content_sources.allow_empty == false then

```

```

210 texio.write_nl("Failed to retrieve " .. macro_name .. " from
    ↳ environment or Git command. Leaving macro undefined.")
-- Do not set the macro, allowing fallback in
    ↳ config/document-settings.sty
else
    texio.write_nl("Failed to retrieve " .. macro_name .. " from
        ↳ environment or Git command. Setting macro to empty.")
    content = ""
end
215 end

-- Ensure content is a string
content = tostring(content):gsub("%s+$", "") -- trim trailing whitespace

220 --[[
    The `content` can contain unprintable characters, like underscores in git
    ↳ branch
    names. Towards this end, use detokenize in the macro itself, which will
    ↳ make all
    characters printable (assigns category code 12). See also:
    https://www.overleaf.com/learn/latex/Articles/An_Introduction_to_LuaTeX_(P|
    ↳ art_2):_Understanding_%5Cdirectlua
225 --]]
local escaped_content = "\\detokenize{" .. content .. "}"

if content then
    texio.write_nl("Setting macro '" .. macro_name .. "' to: '" ..
        ↳ escaped_content .. "'.")
230 -- Set a macro (`\newcommand`) see also:
    ↳ https://tex.stackexchange.com/a/450892/120853
    token.set_macro(macro_name, escaped_content)
end
end
end

```

Rest of this page intentionally left blank.



## A. An appendix chapter

Some appendix content can go here, for example detailed computations.

### A.1. More appendix

The usual sectioning commands keep working, as does page numbering, showcased by the following blind text.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis.

Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

## A.2. Unprocessed data

This is also a good place to put unprocessed data, like [Table A.1](#).

**Table A.1** / A longtblr from the new **tabularray** package. It introduces many new features and concepts and is based on a new interface. For example, this table breaks across pages and has an automatically sized column

Magnitude	Unit	Variable-width text	Math column
10.0	m	Some rather long text that will get broken up and even has a footnote <sup>1</sup>	$x = y$
37.13	G°C	Foo	$H^i = f$
496.111	kbar	Yeet	$a - P = d$
0.23	mm	Baz	$C^u = o$
3.30	nW	Baz	$I + Y = C$
0.6	kΩ	C	$p + n = t$
5.845	bar	B	$Y^H = x$
7.343	nN	Yeet	$Z/z = c$
403.0	GN	Foo	$a^Z = v$
312.81	ncd	Bar	$I - l = g$
0.457	Pa	B	$f + Q = i$
545.773	k°C	Bar	$i + j = W$
76.5	N	Hello World	$o/q = A$
67.20	A	Hello World	$T^e = M$
2.571	Gg	Foo	$O + o = j$
285.542	nPa	A	$o + T = Z$
12.458	V	A	$k/K = b$
668.0	V	B	$p + E = o$
799.923	km	Hello World	$p^j = X$
413.0	n°C	Foo	$j^s = l$
86.64	mg	Yeet	$F^k = C$
50.931	ncd	A	$T^j = M$
0.75	K	Yeet	$X/g = z$
0.184	mK	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$i + k = K$
24.0	GK	Baz	$R + H = a$

Continued on next page

**Table A.1** / A longtblr from the new `tabularray` package. It introduces many new features and concepts and is based on a new interface. For example, this table breaks across pages and has an automatically sized column (Continued)

Magnitude	Unit	Variable-width text	Math column
583.56	nA	B	$R + C = r$
4.92	GPa	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$P/O = z$
196.104	GW	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$u^R = g$
11.2	GV	C	$q + P = R$
60.14	nbar	Bar	$E^B = B$
347.91	mA	Baz	$u^i = M$
563.70	$\Omega$	Baz	$S/y = D$
0.43	mK	Yeet	$w/z = u$
3.851	mK	A	$m/Z = c$
0.74	g	Foo	$b^d = F$
0.98	n $\Omega$	Yeet	$m/w = G$
942.0	Gcd	Hello World	$T^l = S$
426.661	nA	Hello World	$b/n = H$
663.729	GN	Yeet	$j + D = Z$
0.65	ncd	C	$J - s = E$
0.29	bar	B	$f + J = M$
433.72	W	Foo	$s^k = r$
0.96	ng	C	$H/P = m$
51.79	nV	Baz	$b/G = b$
0.744	kW	B	$G^C = X$
0.9	cd	B	$H + i = U$
416.71	GK	Foo	$H^j = I$
447.0	$^{\circ}\text{C}$	Bar	$N + o = K$
0.65	$^{\circ}\text{C}$	Yeet	$K - h = F$
0.954	nPa	Hello World	$z + Y = J$

Continued on next page

**Table A.1** / A longtblr from the new `tabularray` package. It introduces many new features and concepts and is based on a new interface. For example, this table breaks across pages and has an automatically sized column (Continued)

Magnitude	Unit	Variable-width text	Math column
874.234	nW	Baz	$M + i = w$
0.14	V	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$Q + i = v$
4.427	mN	Baz	$B/B = p$
5.4	kg	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$t/A = w$
0.60	cd	B	$n^p = K$
163.0	kA	Yeet	$M + w = b$
53.911	$\Omega$	Yeet	$E - y = a$
61.0	Gcd	Yeet	$T - M = E$
0.686	kcd	Foo	$d^a = f$
6.354	bar	A	$F/L = G$
2.88	k°C	Bar	$M^r = T$
283.457	mg	Hello World	$D - E = u$
490.455	m	Hello World	$S/u = E$
960.1	A	Yeet	$r/M = q$
884.0	bar	B	$w^R = v$
824.9	kW	Baz	$Z - t = T$
91.6	V	Yeet	$E/o = s$
45.0	N	Baz	$Z - m = P$
75.0	nW	C	$Y - U = R$
778.0	mK	Foo	$x^j = L$
0.885	mW	C	$u^S = G$
70.49	mcd	Yeet	$z^R = r$
185.0	kg	Bar	$O + T = A$
4.880	G $\Omega$	A	$F/U = q$
0.78	GK	Bar	$m + C = E$

Continued on next page

**Table A.1** / A `longtblr` from the new `tabularray` package. It introduces many new features and concepts and is based on a new interface. For example, this table breaks across pages and has an automatically sized column (Continued)

Magnitude	Unit	Variable-width text	Math column
712.36	nbar	This is a longer text that will probably span multiple lines in the table because it is overly wide.	$Z + D = p$
37.13	G°C	Foo	$H^i = f$
496.111	kbar	Yeet	$a - P = d$
0.23	mm	Baz	$C^u = o$

<sup>1</sup> Some table footnote.

*Note:* Some general hint regarding the table, which might span multiple lines.

Rest of this page intentionally left blank.



## **B. Another appendix chapter**

More stuff can go into the next appendix chapter, for example algorithms and code.

Rest of this page intentionally left blank.

# Bibliography

- GBV (GBV), Gemeinsamer Bibliotheksverbund (2013): *Gemeinsamer Verbundkatalog (GVK)*. URL: <http://gso.gbv.de/DB=2.1/> (visited on 08/28/2013) (cit. on p. 24).
- BAEHR and KABELAC 2016 BAEHR, Hans Dieter and KABELAC, Stephan (2016): *Thermodynamik: Grundlagen und technische Anwendungen*. 16., aktualisierte Auflage. Lehrbuch. Berlin: Springer Vieweg. 671 pp. ISBN: 978-3-662-49568-1 (cit. on p. 47).
- BIRD et al. 2009 BIRD, Steven et al. (2009): *Natural Language Processing with Python*. 1st ed. Beijing ; Cambridge [Mass.]: O'Reilly. 479 pp. ISBN: 978-0-596-51649-9 (cit. on p. 47).
- DIRAC 1981 DIRAC, Paul Adrien Maurice (1981): *The Principles of Quantum Mechanics*. International Series of Monographs on Physics. Clarendon Press. ISBN: 978-0-19-852011-5 (cit. on pp. 47, 49).
- DUBBEL et al. 2007 DUBBEL, Heinrich et al. (2007): *Taschenbuch Für Den Maschinenbau*. 22nd ed. Berlin Heidelberg New York: Springer. 1 p. ISBN: 978-3-540-49714-1 (cit. on p. 47).
- EINSTEIN 1905 EINSTEIN, Albert (1905): "Zur Elektrodynamik Bewegter Körper. (German) [On the Electrodynamics of Moving Bodies]." In: *Annalen der Physik* 322.10 (10), pp. 891–921. DOI: <http://dx.doi.org/10.1002/andp.19053221004> (cit. on pp. 47, 48).
- GOOSSENS et al. 1993

GOOSSENS, Michel et al. (1993): *The \LaTeX\ Companion*. Reading, Massachusetts: Addison-Wesley (cit. on p. 47).

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION 2017

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (Mar. 2017): *ISO 8217:2017 - Petroleum Products - Fuels (Class F) - Specifications of Marine Fuels*. Standard 8217. ISO/TC 28/SC 4, p. 23. URL: <https://www.iso.org/standard/64247.html> (visited on 10/08/2018).

KARMASIN and RIBING 2010

KARMASIN, Matthias and RIBING, Rainer (2010): *Die Gestaltung wissenschaftlicher Arbeiten*. 5th ed. Wien: Facultas. ISBN: 9783825248222 (cit. on p. 22).

Donald E. KNUTH 1973

KNUTH, Donald E. (1973): *Fundamental Algorithms*. 3rd ed. Vol. 1. 4 vols. The Art of Computer Programming. Reading, Mass.: Addison-Wesley. ISBN: 978-0-201-89683-1 (cit. on p. 47).

Donald Ervin KNUTH 1986

KNUTH, Donald Ervin (1986): *The TeXbook*. Computers & Typesetting A. Reading, Mass: Addison-Wesley. 483 pp. ISBN: 978-0-201-13447-6 (cit. on p. 47).

LPT05

LATEX3 PROJECT TEAM (Nov. 27, 2005): *ETEX 2<sub>ε</sub> font selection*. URL: <http://mirrors.ctan.org/macros/latex/doc/fntguide.pdf> (visited on 08/28/2013) (cit. on pp. 26, 27).

MATHWORKS 2020

MATHWORKS (2020): *Create a Simple Class - MATLAB & Simulink*. URL: [https://www.mathworks.com/help/matlab/matlab\\_oop/create-a-simple-class.html](https://www.mathworks.com/help/matlab/matlab_oop/create-a-simple-class.html) (visited on 04/14/2020) (cit. on p. 92).

MOLLENHAUER and TSCHÖKE 2007

- MOLLENHAUER, Klaus and TSCHÖKE, Helmut (2007): *Handbuch Dieselmotoren*. 3., neubearb. Aufl. VDI-Buch. Berlin: Springer. 702 pp. ISBN: 978-3-540-72164-2.
- PTB07      PHYSIKALISCH-TECHNISCHE BUNDESANSTALT (2007): “Das Internationale Einheitensystem (SI).” In: *PTB-Mitteilungen* 2.2007, pp. 144–180 (cit. on p. 28).
- SESINK 2007      SESINK, Werner (2007): *Einführung in das wissenschaftliche Arbeiten*. 7th ed. München: Oldenbourg Verlag (cit. on p. 22).
- TUB      TUHH UNIVERSITÄTSBIBLIOTHEK (2013): *Katalog der TUB*. URL: <https://katalog.b.tuhh.de/DB=1/LNG=DU/> (visited on 08/28/2013) (cit. on p. 24).
- VOSS 2011      Voss, Rödiger (2011): *Wissenschaftliches Arbeiten*. 2nd ed. Konstanz: UVK Verlagsgesellschaft. ISBN: 978-3-8252-8483-1 (cit. on pp. 21, 22, 25).
- WIKIPEDIA CONTRIBUTORS 2021      WIKIPEDIA CONTRIBUTORS (Jan. 5, 2021): *Modelica*. In: *Wikipedia*. Ed. by WIKIPEDIA CONTRIBUTORS. URL: <https://en.wikipedia.org/w/index.php?title=Modelica&oldid=998516587> (visited on 02/19/2021) (cit. on p. 95).

## Further Reading

*The following references were used in this work but not cited in the text body; they are provided here as-is.*

- EXAMPLE, Jane (2024): *Demonstration Citation Entry*. Placeholder reference used for Omni-LaTeX citation examples.
- McKINNEY, Wes (2018): *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. Second edition. Sebastopol, California: O’Reilly Media, Inc. 524 pp. ISBN: 978-1-4919-5766-0.

RAMALHO, Luciano (2015): *Fluent Python*. First edition. Sebastopol, CA: O'Reilly. 743 pp.  
ISBN: 978-1-4919-4600-8.

# Index

Terms	Names
	MACH
	MOLLIER
LaTeX package	REYNOLDS
TikZ, xi, xii, 60, 68, 69, 71, 74, 77, 80–83	TUFTE, 73