

docART



A Python/Lua(\LaTeX) Utility and \LaTeX Class
for Semi-automated Documentation
of Projects involving Hardware and/or Software.

Authors: Martin Stimpfl, Daniel Sacré


– Handbook –

docART Utility, alpha-2022-04-30
Document Revision 0

License: GNU General Public License (GPLv3)



Contents

1. Preface: T_EX, L^AT_EX, LuaL^AT_EX and the Documentation Problem in IT	4
2. Aim and Scope of the docART Utility	5
3. Legal Disclaimer	6
4. Licensing (Quick Overview)	7
5. Installation & Configuration (Quick Guide)	8
6. Configuration	9
6.1. docART	9
6.2. TeXstudio	9
7. Usage	11
7.1. Terminal/Command Line	11
7.2. TeXstudio	12
8. docART: Basic Functionality	13
8.1. Highlight Boxes	13
8.2. Icons () in Text	14
8.3. Figures	16
8.4. Tables	20
8.4.1. General Tables	20
8.4.2. Specific Tables	24
8.5. Listings of Programs and Files	26
8.5.1. Inline	26
8.5.2. Inblock	26
8.5.3. Changing the Listing Syntax Highlighting	32
8.5.4. Listings: Summary	35
8.6. References and Labels	36
8.6.1. Usage	36
8.6.2. Recommended Label Naming Convention	37
9. Adapting the Default docART Theme	38
9.1. Project Details	38
9.2. pdf Metadata	39
9.3. Page Setup	40
9.3.1. Title Page	40
9.3.2. Margins and Type Area	41
9.3.3. Page Header and Footer	41
9.4. Fonts	42



docART Utility Handbook



9.5. Colors	43
10. Known Bugs	44
Appendix	45
A. Contributors	46
B. Licensing	47
B.1. Mandatory Products	47
B.1.1. \LaTeX -Backend	47
B.1.2. Python	48
B.1.3. Lua Hash Library	48
B.1.4. docART Utility	49
B.2. Optional Products	53
B.2.1. Fonts	53
B.2.2. TeXstudio	53
C. Installation	54
C.1. Mandatory Steps	54
C.1.1. Windows	56
C.1.2. Linux	56
C.1.3. Mac OS	56
C.2. Optional	57
C.2.1. TeXstudio	57
D. Listing of Source Code	59
D.1. docART Utility	59
D.1.1. \LaTeX Class	59
D.1.2. Python Scripts	62
D.1.3. Lua Scripts	64
D.2. docART Utility Handbook	74
D.2.1. docART Theme Files	74
D.2.2. User Settings	81
D.2.3. Content Examples	86



1. Preface: T_EX, L^AT_EX, LuaL^AT_EX and the Documentation Problem in IT

Typesetting has been a complicated subject for centuries. In science, especially mathematics, many symbols are unique to the subject and bear little to no resemblance with standard Latin letters. This meant for typesetting a mathematical text, special typewriters were mandatory. With the advent of computers, the challenge of how to interface with them for typesetting purposes, format (“What You See Is What You Get” (WYSIWYG) versus “What You See Is What You Want” (WYSIWYW)) and in the end print text shifted the complexity to an even higher level.

A solution for mathematicians was T_EX, developed by Donald Knuth in the late 1970s. It shines especially when typesetting complex mathematical formulae, but due to its versatility and multilingual/multisymbol (Greek, Sanskrit, etc.) support quickly spread into other domains. Up to this date, T_EX is considered to be one of the most sophisticated typographical systems. [1]

One downside of T_EX was its complexity; or, to put it into other words, its user unfriendliness. In the early 1980s, Leslie Lamport [2] created L^AT_EX, which is (from a very oversimplified point of view) a T_EX macro collection focusing on user friendliness and ease of use. Over the years, it evolved further and further. Additional functionality has been implemented with the many derivatives like x_eL^AT_EX and LuaL^AT_EX, the latter one adding direct support for the Lua scripting language.

The authors of the utility, both with a background in physics and programming, asked themselves the question, why L^AT_EX is not more widely used outside of science. The consistent template generation, high possible level of automation and its OpenSource license (T_EX is in the public domain and therefor free of charge) should attract some interest. Its usefulness in computer science should also apply to the IT sector. Additionally, programmers should not be afraid of writing (a document in) code, right?

It turns out that most of the employees responsible for the documentations are not necessarily programmers themselves. Creating a document outside of a WYSIWYG application, where it is necessary to write “code”, frightens them, which in turn means they refuse vehemently a WYSIWYW solution if it gets proposed. The other scenario, even when programmers are responsible, is laziness: “We have always been doing it like this... It always worked somehow... Everybody uses **** (insert here the most famous word processor), why shouldn’t we, too?”... All the merge conflicts (multiple people working on the same document, overwriting the changes of the other; proprietary binary files of word processors not version controllable, ...) as well as inconsistency between the code shown in the documentation and the actual product that shipped (last-minute changes before the release not copied into the documentation, etc.) and the myriad of other not easily fixable problems are ignored. Not to mention the typesetting sins that many WYSIWYG programs create or let you get away with...

[1] <https://en.wikipedia.org/wiki/TeX>, last visited 2022-03-31

[2] <https://en.wikipedia.org/wiki/LaTeX>, last visited 2022-03-31



2. Aim and Scope of the docART Utility

Like mentioned in the preface 1, many people seem to be scared to create documents with WYSIWYW-applications. For a newcomer, the syntax and many options can be overwhelming at first. In addition, especially \LaTeX has a steep learning curve. The authors of this utility wish to facilitate the usage of \LaTeX for newcomers by providing a collection of easy-to-use wrapper macros without sacrificing any of the \LaTeX -functionality and flexibility. More advanced users could still write any other valid \LaTeX code without having to deal with the tedious task of manually typesetting tables and listings. Besides that, the docART wants to make the most important feature of \LaTeX ((almost) everything gets recreated from scratch during a compilation) more accessible: Is an included image altered from an external source, it will be updated in the document during the next \LaTeX compilation. Same holds for imported source code, tables, etc.. As a bonus, docART provides additional parsing scripts which run in the background to improve compilation efficiency as well as user friendliness.

To put it in a nutshell, docART continues the honored \TeX tradition of creating macro collections for better usability and adds some additional parsing scripts in the backend. It is targeted for \LaTeX beginners and professionals alike. The authors see as main use cases for docART alpha-2022-04-30:

- Prototype development (Hard- and Software),
- Software Development Kit (SDK) documentation,
- User manuals/command reference for machines and scientific equipment,
- Penetration testing documentation.

Since the docART Utility at its core is a (wrapper) macro collection with some additional scripts, some of the content of this handbook describes functionality provided by other \LaTeX packages. To give the reader a better overview what macro is defined by the docART Utility and which is inherited from another package, the macros defined by the docART Utility are highlighted throughout this document in **orange**, whereas the ones which stem from another source are highlighted in **blue**. Furthermore, (with few exceptions like general, not necessarily docART Utility specific macros $\text{\code{\productName}}$, $\text{\code{\productVersion}}$, etc.) all environments and macros defined by the docART Utility start with the prefix “da” (e.g. **daInfoBox**, **daIcon**, etc.).



3. Legal Disclaimer

- Trademarks are used throughout the document to describe software, hardware and functionality. None of the authors is holding one of the trademarks mentioned nor is affiliated with any of the trademark holders. All trademarks belong to their respective owners. Before reusing the trademarks in another product, the reader is responsible to check whether the intended usage is prohibited. The authors of this documentation deny any legal responsibility for damages caused by unverified re-usage.
- All licenses have been verified to the best knowledge of the authors during the writing of the document. The authors cannot guarantee that the licensing status of fonts, programs or tools etc. change over time. The reader is responsible for verifying if the intended use case is permitted by the license. The authors of this documentation deny any legal responsibility for damages caused by unverified re-usage.
- All web hyperlinks have been verified during the writing of the document. The authors of this documentation deny any legal responsibility for correctness of the information presented on the linked web pages or damages caused by visiting the web pages.
- Any suggestions and advises regarding good practices, typesetting rules or maximizing efficiency are personal opinions of the authors based upon their scientific typesetting training and experience. The authors deny any responsibility if the product you try to produce with docART does not look good or is not being approved by (typesetting) authorities.
- The software is provided “as is”, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the software or the use or other dealings in the software.



4. Licensing (Quick Overview)

Since the docART Utility is built-upon many other OpenSource projects, its Backend is comprised of products with many different licenses. The following licenses are present in the products that are essential for a working docART Utility:

MIT, GNU Public License (GPL), LaTeX Project Public License (LPPL),
The Python Software Foundation (PSF) License Agreement

The fonts which the docART Utility ships are released under MIT, Bitstream Vera License and GUST License. However, if required, the fonts can be easily swapped (see section 9). Since many of the Backend products are bundles, their individual sub-modules might have different licenses. The agreed upon standard is that the redistribution is possible under the bundle's redistribution license. All the docART Utility's end user cases are in agreement with the products' individual licenses, so the usage of the docART Utility itself is safe (except re-bundling/-distribution). To help the end user, the authors provide a detailed listing of all the products and their respective licensing in appendix B. However, the authors do not take any responsibility that the information is correct. If in doubt, the end user is responsible to check whether the intended usage is prohibited. The Free Software Foundation's (FSF) definition and the Debian Free Software Guidelines can be helpful for deciding edge cases.

The docART Utility itself is licensed under GNU General Public License (GPLv3). This includes all \LaTeX -, Python-, and Lua-files as well as the docART specific dcmts-files and the documentation (pdf as well as source). The only exception is the docART logo, which is released under Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0).

If you are re-using some part of the docART Utility's source code, please give credit according to the GPLv3 rules as follows:

Martin Stimpfl, Daniel Sacré: docART Utility, alpha-2022-04-30,
License: GNU General Public License (GPLv3).



Since a \LaTeX file is a mixture of source code and content, there is no clear definition of when giving credit/re-publishing the source code is necessary. The authors therefore decided the following: The (\LaTeX) source code written by the user contains more data than actual docART Utility source code. Consequently, it has to be treated as data, which is belonging to its creator, the end user. All the docART Utility does is to process the data into a formatted output as binary pdf, thus the pdf output can be distributed under any license. As long as the end user did not do substantial changes to the docART Utility source code, it does not have to be published. For details about what substantial source code changes are, please see the examples in appendix B.1.4, page 50 and following.



5. Installation & Configuration (Quick Guide)



The docART Utility uses shell-escape to provide its functionality. Please check BEFORE installing the utility, whether your IT security policy allows this.

To install the docART Utility, please do the following:

Step 1: Install a pdf Reader.

Required for viewing the pdf output.

Step 2: Install Python 3.

Recommended:

Download from <https://www.python.org/downloads/> (Windows),
package manager (Linux)

Step 3: Install the \LaTeX -Backend.

Recommended: MikTeX (Windows), texlive (Linux)

This step has to be done BEFORE installing optional \LaTeX components like TeXstudio.

Step 4: Install the Optional Components (TeXstudio, etc.).

Step 5: Download and Unpack the docART Utility.

Download current-release branch from
<https://github.com/d-sacre/docart-documentation-utility>

Step 6: Configuration (see section 6):

Step 6.1: docART: Rename the main \LaTeX -file and customize the theme.

Optional: Delete the Handbook source code.

Step 6.2: TeXstudio (optional): Enable shell-escape.

A detailed installation guide can be found in appendix C.



6. Configuration

6.1. docART

After downloading and unpacking/cloning the docART github repository, the docART Utility is ready for use with the terminal/command line. However, one should do some clean-up and adapt it to ones' needs. To do so, open a file browser or terminal/command line and navigate to the folder where docART is located.

- Step 1:** In most cases it makes sense to delete the complete documentation source code in `./docart-utility/documentation/handbook_src/`. Alternatively one can also simply delete the entire directory `./docart-utility/documentation/`, which also removes the compiled pdf of the documentation.
- Step 2:** If one is not planning to use `git` for version control, one can delete the `./.git/`-folder and the `.gitignore`-file. Otherwise, one should re-initialize the repository and adapt the `.gitignore`-file to ones' requirements.
- Step 3:** One should rename the `docart_mwe.tex` to a name representative for the intended use case, as the name of the compiled pdf file will be derived from the one of the main \LaTeX file that gets compiled.
- Step 4:** Adjust the project setting files located at `./settings/` (for details: see section 9). This is not necessary for the functionality and can be done at a later date. As long as these settings are not changed, the default values for e.g. `\productName`, pdf author etc. will be used.

6.2. TeXstudio

For the usage with the docART Utility, the compilation options in TeXstudio have to be changed to provide shell escape functionality.



A TeXstudio compilation command containing shell escape is considered dangerous, as this setting is permanently present, even when not in use. This might enable attackers to execute malicious code. Please check whether permanent shell escape is compliant with your IT Security Policy before proceeding.

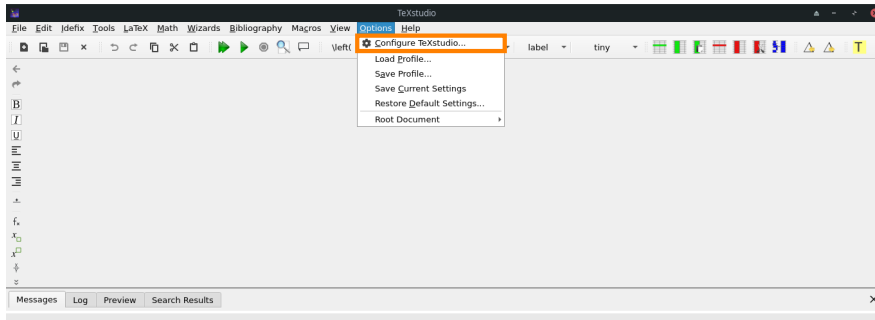


docART Utility Handbook

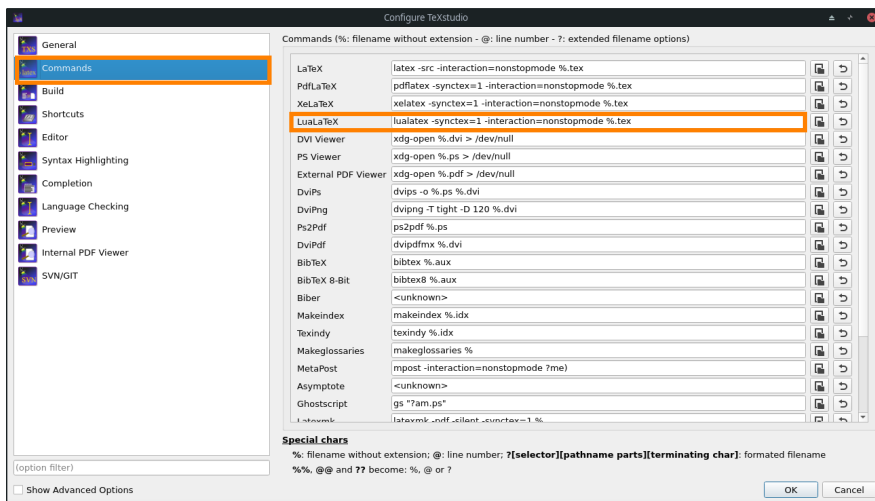


Step 1: Start TeXstudio.

Step 2: Under “Options” select “Configure TeXstudio...”.



Step 3: Navigate to the “Command”-tab.



Replace the standard “LuaLaTeX”-entry

```
lualatex -synctex=1 -interaction=nonstopmode %.tex
```

with:

```
lualatex --shell-escape -synctex=1 -interaction=nonstopmode %.tex
```

Step 4: Confirm the changes by clicking on the “OK”-button. The changes will be applied and the configuration window will close.



7. Usage

7.1. Terminal/Command Line

To use the docART Utility in the terminal/command line, proceed as follows:

Step 1: Open a terminal/command line and set via `cd` the working directory to the folder where your docART project files are located.

Step 2: Run the command

```
lualatex --shell-escape LATEXFILENAME.tex
```

where you replace `LATEXFILENAME` with the name of the \LaTeX -file located in the root directory of your project folder.



The \LaTeX command has to be run from a shell with a working directory identical to the root folder of your docART project. Even providing an absolute path like

```
lualatex --shell-escape /ABS/PATH/T0/LATEXFILE.tex
```

leads to the problem that the docART class-file cannot be loaded.



- Certain elements (e. g. table of contents, references, tables etc.) need at least two compilations to be displayed properly. Therefore the authors recommend to always compile twice by running `lualatex --shell-escape LATEXFILENAME.tex` twice in succession.
- Running two compilations via

```
lualatex --shell-escape LATEXFILENAME.tex &&  
lualatex --shell-escape LATEXFILENAME.tex
```

is not recommended. If a compilation error should occur, the compilation would restart and throwing the same error another time. This is especially not favorable if the compilation duration is long.
- More complex compilation pipelines for example like required for creating a bibliography can be achieved with a `make-file`.



7.2. TeXstudio

To use the docART Utility with TeXstudio, proceed as follows:

- Step 1:** Open the \LaTeX -file located in the root directory of your docART project folder with TeXstudio.
- Step 2:** To compile the \LaTeX source code and view the pdf output in the internal pdf viewer, press either F5 on your keyboard or the “Build & View”-button in the GUI (see figure 7.1).

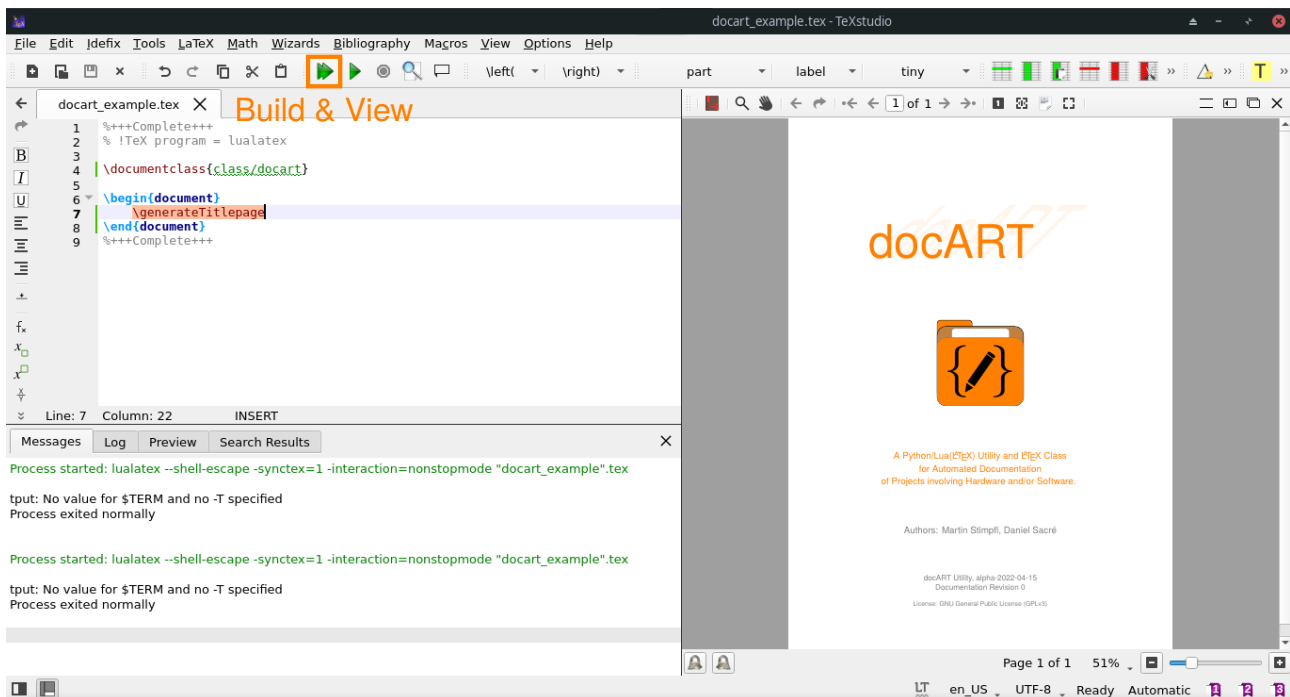


Figure 7.1: Screenshot of the compilation of a file in TeXstudio. \LaTeX source code (top left), Messages/Log/Preview/Search Results (bottom left) and a view of the compiled pdf (right). The “Build & View”-button (top) is highlighted in orange.



TeXstudio automatically determines the minimum necessary amount of compilations to produce a correct pdf output (including a correctly updated table of contents, tables, references, etc.). In contrary to the terminal/command line usage described in section 7.1, the users do not have to determine this number by themselves. A disadvantage of using TeXstudio over the terminal/-command line is that the compilation takes longer and the error messages might not be as precise/immediate.



8. docART: Basic Functionality

8.1. Highlight Boxes

Sometimes it is necessary to highlight the importance of information. Changing the font type to bold or italics is not the best way. On the one hand, it breaks the reading flow and on the other hand font changes can be too subtle if the reader is just scanning the document very rapidly.



This is a box designed to indicate a warning. Its design is by purpose very minimalistic, but can be altered to suit any needs. Additionally, it does not use the complete line width to provide a clear visual separation between continuous text and the boxes.

These boxes should only contain very little text and used sparsely; otherwise the highlighting effect wears off. By design, these boxes do not provide pagebreak functionality, which is intentional to force the user to fill these boxes with only the necessary content.

The highlight box above was created using the following \LaTeX code:

```
% Generating a warning box.
```

```
\begin{daWarningBox}
```

```
This is a box designed to indicate a warning. Its design is by  
purpose very minimalistic, but can be altered to suit any needs.  
Additionally, it does not use the complete line width to provide a  
clear visual separation between continuous text and the boxes.
```

```
\end{daWarningBox}
```

Listing 8.1: The \LaTeX code required for generating a warning box.

In alpha-2022-04-30, docART provides a second custom environment called **daInfoBox**. Replacing **daWarningBox** with **daInfoBox** in the code snippet above, leads to the following output:



This is a box designed to indicate important information. Its design is by purpose very minimalistic, but can be altered to suit any needs. Additionally, it does not use the complete line width to provide a clear visual separation between continuous text and the boxes.



8.2. Icons () in Text

In comparison to many other word processors, the docART Backend allows inclusion of custom icons into the continuous text. It also works for headings, tables, captions and lists. The icons can be either images (png, jpg, pdf), TikZ-Elements and more. There is also a native scalability with font size, as it is demonstrated below with some of the default icons the docART Utility ships with.



For version alpha-2022-04-30, the authors do not recommend the user to create custom icons and therefore do not provide a how-to-guide. The reason being that in one of the next releases, substantial changes will be made to the icons Backend, which should make the icon creation by the end user way easier and more consistent.

This is a test normal sized text with an  icon within the text.







This is is a normal sized text with an icon at the end .

This is a large text with an  icon within the text.






This is a Large text with an  icon within the text.

This is a LARGE text with an  icon within the text.

Listing 8.2: \LaTeX code for using icons in the text.

```
\section{Icons () in Text}
  This is a test normal sized text with an  icon within the text
  .\[\[0.125cm]
  This is is a normal sized text with an icon at the end .\[\[0.125
  cm]
  {\large This is a large text with an  icon within the text
  .}\[\[0.125cm]
  {\Large This is a Large text with an  icon within the text
  .}\[\[0.125cm]
  {\LARGE This is a LARGE text with an  icon within the text
  .}\[\[0.125cm]
```

The  comes in handy if a new font/typeface should be assessed.







For everyday use, docART provides a selection of default icons. The icons can be accessed by calling the macro `{ICONNAME}`. For example, `{warning}` generates an exclamation mark within a yellow triangle . Icons with names containing “var” can be scaled manually. For example, `[10pt]{varwrench}` generates a wrench  of width 10 pt. The table 8.1 lists all the currently available icons in docART alpha-2022-04-30.



docART Utility Handbook



Table 8.1: Overview of available icons in docART alpha-2022-04-30.

icon	ICONNAME	description
	warning	exclamation mark within yellow triangle
	info	letter "i" within blue circle
	varinfo	letter "i" within yellow circle
	varwrench	a symbolic representation of a wrench
	varpaperclip	a symbolic representation of a paperclip
	docART	icon version of the docART logo



8.3. Figures

In docART, the minimum requirement for creating a figure is to specify the path to the image file(s) (supported formats: png, jpeg, pdf, eps (not recommended by the authors)); the correct layouting/formatting is handled automatically. For example, the single image with plain formatting



was created by calling

```
\daFigureDefaultCaptionOff{./pictures/testpicture.png;}
```

In alpha-2022-04-30, this macro call creates a plain figure without a caption and a default image width of 0.495\linewidth; changing the width or specifying the height is not possible in this version of the docART Utility.

To generate a figure with a caption below, replace the `\daFigureDefaultCaptionOff` with `\daFigureDefaultCaptionBelow` and add the required information for the caption,

```
\daFigureDefaultCaptionBelow{./pictures/testpicture.png;}{%  
  fig:bf:figures:example:one-image:no-grid-option:caption-below%  
}{%  
  Same figure with a caption below.%  
}
```



Figure 8.2: Same figure with a caption below.

which generates the figure 8.2. The general syntax for figures with a single image and captions is `\daFigureDefaultCaptionBelow{FILEPATH;}{LABEL}{CAPTION}`. For a detailed explanation what the LABEL property can be used for, please refer to section 8.6.



To load multiple images with one shared caption, simply specify all of the file paths separated by a semicolon: `\daFigureDefaultCaptionBelow{FILEPATH1;FILEPATH2;}{LABEL}{CAPTION}`. By default, the images will be arranged into two columns and scaled to 0.495\linewidth .



Figure 8.3: Loading two images with a caption below.

The figure 8.3 was generated by the following code:

```
\daFigureDefaultCaptionBelow{
  ./pictures/testpicture.png;
  ./pictures/testpicture.png;
}{fig:bf:figures:example:two-images:no-grid-option:caption-below}{%
  Loading two images with a caption below.%
}
```

For better readability, the lines may be broken after the semicolon.



In contrast to the recommended \LaTeX convention to prevent issues occurring by line breaks in macro arguments by terminating the lines with a comment `%`, this has to be omitted for the (first) mandatory argument as this would lead to parsing issues and an aborted compilation.

A layout with more than two columns is possible by specifying the `grid` option (see figure 8.4).

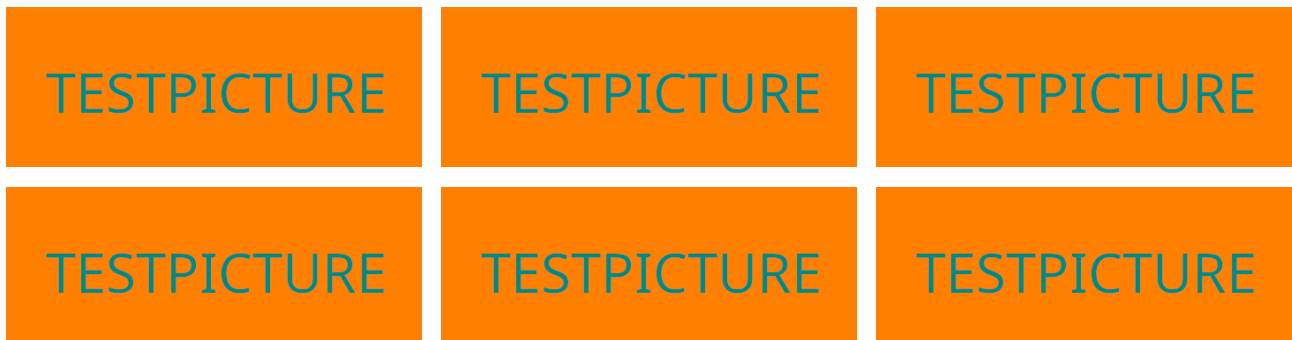


Figure 8.4: Example how to use the `grid=2x3` option to force a three column layout with two rows total. Specifying more images (seven instead of six) results in the excess files being ignored.



The figure 8.4 was created by the command shown in listing 8.3.

```
\daFigureDefaultCaptionBelow[grid=2x3]{  
  ./pictures/testpicture.png;  
  ./pictures/testpicture.png;  
  ./pictures/testpicture.png;  
  ./pictures/testpicture.png;  
  ./pictures/testpicture.png;  
  ./pictures/testpicture.png;  
  ./pictures/testpicture.png;  
}{fig:bf:figures:example:usage-grid-option-and-over-specify}{%  
  Example how to use the \lstinline$grid=2x3$ option to force a three column  
  layout with two rows total. Specifying more images (seven instead of six)  
  results in the excess files being ignored.%  
}
```

Listing 8.3: The command used to generate figure 8.4.

The option `grid=NxM` creates a figure layout with `N` rows and `M` columns (e.g. figure 8.4: two rows, three columns). Additionally, listing 8.3 shows the way how more specified file paths (seven in the example) as available image slots (maximum amount of images: `ROWS` times `COLS`, e.g. figure 8.4: $2 \cdot 3 = 6$) are handled. Only the amount of images that can be typeset are processed, starting from the top of the list to the bottom, ignoring superfluous file paths. A similar procedure is applied for the case of missing file paths (see figure 8.5 and figure 8.6).

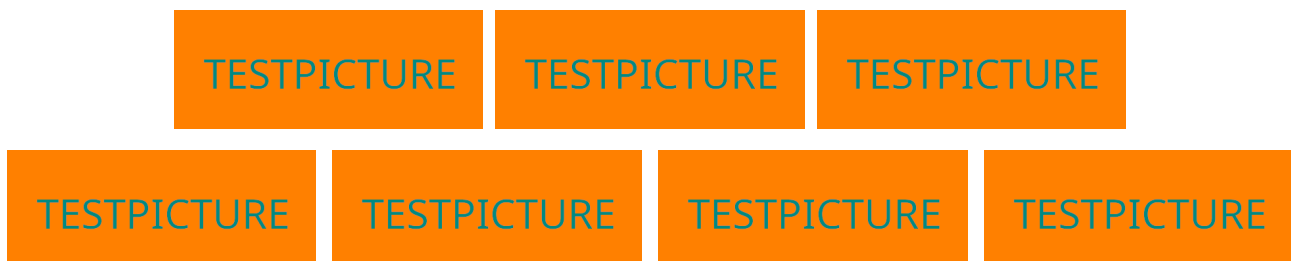


Figure 8.5: Example how specifying less images (seven instead of eight) than would fit in a grid with even number of columns (2x4 grid) is handled.

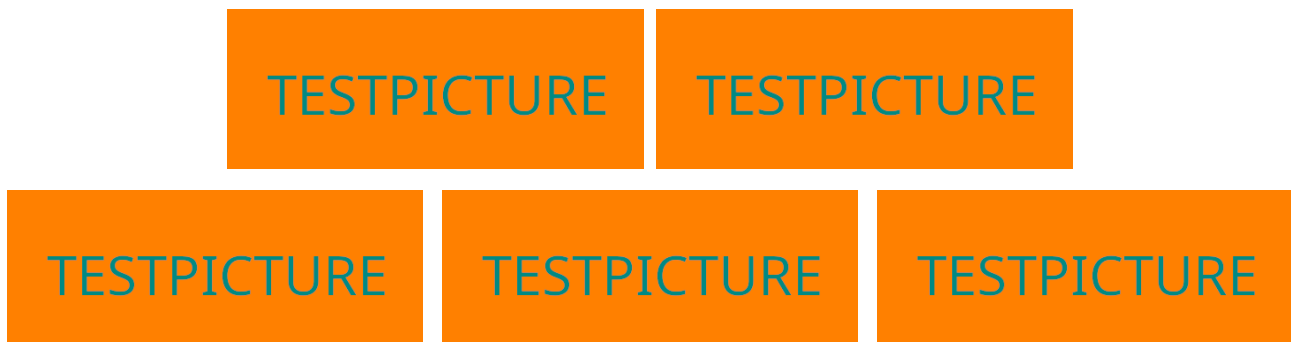


Figure 8.6: Example how specifying less images (five instead of six) than would fit in a grid with odd number of columns (2x3 grid) is handled.



docART Utility Handbook



The cases depicted in figure 8.5 and figure 8.6 showcase the behavior of the docART Utility when too few file paths are specified for a layout with even/odd number of columns, respectively. All the specified images will be typeset; instead of setting a placeholder image, the first row will contain one element less than the other rows. For a better visual impression, the first row will also be centered differently.

In contrast to the single and two column layouts shown in figure 8.2 and figure 8.3, respectively, the image width in the three and four column layouts of figure 8.4, figure 8.5 and figure 8.6 is reduced to accommodate the additional images without exceeding the type area. Although it is technically possible, the authors would not recommend to use more than four columns, since otherwise the images will get too small. Similar to the columns, the number of rows has no technical limit except the page height. Due to the fact that the height of the individual images is not adjustable and the visual pollution if too many images/much content are/is on a single page, the authors recommend not to use more than six rows (depending on the height of the images).



In \LaTeX , a single figure object cannot span multiple pages. Please take this into account when designing the individual images and while you are creating your document structure. If necessary, split the images into two or more independent figure objects.



8.4. Tables

8.4.1. General Tables

In alpha-2022-04-30, docART provides the option to typeset tables directly from a comma separated csv-file. For example, the csv-file for the table with the Python module license data in section B.1.2 is shown in listing 8.4.

```
module,PSL,license
os,yes,PSF
sys,yes,PSF
re,yes,PSF
time,yes,PSF
hashlib,yes,PSF
xml.etree.ElementTree,yes,PSF
csv,yes,PSF
```

Listing 8.4: Contents of the text file `python-module-dependencies_license.csv`, which is parsed by \LaTeX to generate the following two tables.

By calling `\daTableDefaultFancyCaptionOff{FILEPATH}`, \LaTeX typesets the csv-file found at `FILEPATH` as a formatted table.

module	PSL	license
os	yes	PSF
sys	yes	PSF
re	yes	PSF
time	yes	PSF
hashlib	yes	PSF
xml.etree.ElementTree	yes	PSF
csv	yes	PSF

The first line in the csv-file is treated as the table header, which is typeset in bold and separated by a thin horizontal line from the rest of the table. All column contents are by default aligned to the left. Since (especially vertical) lines in tables are from a digital typesetting point considered as problematic, some visual content separation is achieved by highlighting every odd data row in gray.



- Please note that if the data in the csv-file needs to contain the following \LaTeX special characters, they have to be escaped prior to loading the file with \LaTeX : `\`, `$`, `&`, `{`, `}`, `%`, `~`. Otherwise the compilation will halt and not produce a valid pdf-output.
- The entirety of the csv-file may neither contain comments nor different amount of columns (especially first row is critical).



module	PSL	license
os	yes	PSF
sys	yes	PSF
re	yes	PSF
time	yes	PSF
hashlib	yes	PSF
xml.etree.ElementTree	yes	PSF
csv	yes	PSF

Table 8.3: The same licensing table as before, but this time with a caption at the bottom.

A caption can be added to the bottom of the table by using

`\daTableDefaultFancyCaptionBelow{FILEPATH}{LABEL}{CAPTION}`.

For reference what the LABEL option can be used for, see section 8.6. To generate table 8.3, the \LaTeX code in listing 8.5 has been used.

```
\daTableDefaultFancyCaptionBelow{%
./tables/docART_python-module-dependencies_license.csv%
}{%
tab:bf:defaultTableWithCaption%
}{%
The same licensing table as before, but this time with a caption
at the bottom.%
}
```

Listing 8.5: \LaTeX code to generate table 8.3.

All the docART table macros are designed such that they can process any desired number of columns, for each of which the alignment can be specified optionally. Additionally, the automatic page break features allows for unlimited numbers of rows with any kind of data string (\LaTeX macros will be executed). These features are demonstrated in table 8.4.

Table 8.4: A fictitious server log table generated with automated page break and executed \LaTeX macros.

date	time	time zone	event
2019/01/01	00:00	CET	server installation finished
2019/01/01	00:05	CET	server successfully booted
2019/01/01	00:06	CET	starting xyz daemon
2019/01/01	00:10	CET	admin login
2019/12/31	23:58	CET	sudo reboot
2019/12/31	23:59	CET	server reboot
2020/01/01	00:00	CET	server back online

Table continued on next page.



Table 8.4 continued from previous page.

date	time	time zone	event
2020/01/01	00:06	CET	starting xyz daemon
2020/01/01	00:10	CET	admin login
2020/12/31	23:58	CET	sudo reboot
2020/12/31	23:59	CET	server reboot
2021/01/01	00:00	CET	server back online
2021/01/01	00:06	CET	starting xyz daemon
2021/01/01	00:10	CET	admin login
2021/12/31	23:58	CET	sudo reboot
2021/12/31	23:59	CET	server reboot

The table 8.4 is generated from the same csv-file shown in listing 8.6.

Listing 8.6: Contents of the csv-file `docART_table_auto-pagebreak-demo.csv`, which is parsed by \LaTeX to generate the table 8.4.

```
date,time,time zone, event
\textcolor{docartTurquoise}{2019/01/01}, \textcolor{docartTurquoise}{00:00}, \
\textcolor{docartTurquoise}{CET}, \textcolor{docartTurquoise}{server
installation finished}
2019/01/01, 00:05, CET, server successfully booted
2019/01/01, 00:06, CET, starting \lstinline$xyz daemon$
2019/01/01, 00:10, CET, admin login
2019/12/31, 23:58, CET, \lstinline$sudo reboot$
2019/12/31, 23:59, CET, server reboot
\textcolor{docartTurquoise}{2020/01/01}, \textcolor{docartTurquoise}{00:00}, \
\textcolor{docartTurquoise}{CET}, \textcolor{docartTurquoise}{server back
online}
2020/01/01, 00:06, CET, starting \lstinline$xyz daemon$
2020/01/01, 00:10, CET, admin login
2020/12/31, 23:58, CET, \lstinline$sudo reboot$
2020/12/31, 23:59, CET, server reboot
\textcolor{docartTurquoise}{2021/01/01}, \textcolor{docartTurquoise}{00:00}, \
\textcolor{docartTurquoise}{CET}, \textcolor{docartTurquoise}{server back
online}
2021/01/01, 00:06, CET, starting \lstinline$xyz daemon$
2021/01/01, 00:10, CET, admin login
2021/12/31, 23:58, CET, \lstinline$sudo reboot$
2021/12/31, 23:59, CET, server reboot
```

Utilizing `\daTableDefaultFancyCaptionAbove[COLALIGN]{FILEPATH}{LABEL}{CAPTION}` for the generation of table 8.4, puts the caption at the beginning of the table. This placement is recommended for tables spanning multiple pages, since having to search for the caption is not very convenient. For better continuity, a reminder caption with the table number will be added at the top of each new page. In the following, the settings will be explored in detail on the basis of listing 8.7.



```
\daTableDefaultFancyCaptionAbove[lcc1]{%  
  ./tables/docART_table_auto-pagebreak-demo.csv%  
}{tab:bf:pagebreak:tables:caption-above}{%  
  A fictitious server log table generated with automated page break and  
  executed \LaTeX~macros.%  
}
```

Listing 8.7: \LaTeX code to generate table 8.4.

For the example shown in listing 8.7, the macro `\daTableDefaultFancyCaptionAbove` forces the caption to be typeset before the table. The optional COLALIGN argument of `lcc1` aligns columns 1 and 4 to the left, whereas columns 2 and 3 get centered. Valid column alignment specifiers are `l` (“left”), `c` (“center”) and `r` (“right”). Being an optional argument, there can be less column alignment specifiers provided than the number of actual columns of the table (with zero being a valid option). The column alignment specifiers will be applied onto the table columns from left to right until no more specifiers are available. The rest of the columns will then be automatically set to align left.

The mandatory arguments are self-explanatory: FILEPATH specifies the path to the csv-file that should be typeset; the LABEL allows to reference the table later on (see section 8.6), whereas CAPTION provides the content of the table caption.

All the in docART, alpha-2022-04-30 available table typesetting macros are summarized below.

Table typesetting macros – summary:

<code>\daTableDefaultFancyCaptionOff</code>	[COLALIGN]{FILEPATH}
<code>\daTableDefaultFancyCaptionBelow</code>	[COLALIGN]{FILEPATH}{LABEL}{CAPTION}
<code>\daTableDefaultFancyCaptionAbove</code>	[COLALIGN]{FILEPATH}{LABEL}{CAPTION}

Valid options for COLALIGN: `l` (“left”), `c` (“center”), `r` (“right”).



- In docART, alpha-2022-04-30 the table page break, reminder captions and \LaTeX macro execution are completely automatic and cannot be influenced by the user.
- Some \LaTeX macros will not execute (properly) within a table. Sometimes special table-friendly versions of the macros have to be used instead.



8.4.2. Specific Tables



This is an experimental feature in version alpha-2022-04-30. The syntax and output might change completely. It is also possible that this feature will be dropped in future releases.

There is also the option to derive custom tailored tables from the default one. For example, table 8.6 shows how an automatic highlighted listing of data types could look like.

data type	description
<code>nullptr</code>	Nullpointer
<code>void</code>	Void
<code>bool</code>	Boolean
<code>char</code>	A single character
<code>wchar_t</code>	Wide character
<code>short</code>	A signed integer of at least 16 bit
<code>int</code>	A signed integer of at least 16 bit
<code>long int</code>	A signed integer of at least 32 bit
<code>long long int</code>	A signed integer of at least 64 bit
<code>unsigned int</code>	An unsigned integer of at least 16 bit
<code>float</code>	A singel precision floting point number
<code>double</code>	A double precision floting point number
<code>std::string</code>	A standard string
<code>std::vector<uint8_t></code>	A standard vector of 8 bit integers
<code>std::list<std::string></code>	A standard list of standard strings

Table 8.6: A simple table containing C++ code snippets.

The table 8.6 is created with

```
\lstset{style=C++}
\daTableCodeDescriptionCaptionBelow{%
  ./tables/docART_code-listing-table_c++-example.csv%
}{tab:bf:tables:function-parameters}{%
  A simple table containing C++ code snippets.%
}
```

from the csv-file shown in listing 8.8.



In version alpha-2022-04-30, the backend for the generation of a table with `\daTableCodeDescriptionCaptionBelow` is different. Therefore, the output and behavior are slightly altered.



```
nullptr, Nullpointer  
void, Void  
bool, Boolean  
char, A single character  
wchar_t, Wide character  
short, A signed integer of at least 16 bit  
int, A signed integer of at least 16 bit  
long int, A signed integer of at least 32 bit  
long long int, A signed integer of at least 64 bit  
unsigned int, An unsigned integer of at least 16 bit  
float, A single precision floating point number  
double, A double precision floating point number  
std::string, A standard string  
std::vector<uint8_t>, A standard vector of 8 bit integers  
std::list<std::string>, A standard list of standard strings
```

Listing 8.8: Contents of the csv-file `./tables/docART_code-listing-table_c++-example.csv`, which is parsed by \LaTeX to generate the table 8.6.

For a detailed guide how to change the syntax highlighting, see section 8.5.3.



8.5. Listings of Programs and Files

The docART Utility uses the `listings` package to provide the syntax highlighting functionality. The source code parsing is done with a Python script provided by the docART Utility developers.

8.5.1. Inline

For including code snippets within the continuous text, the docART Utility provides the macro `\daListingInline[OPTIONS]{CONTENT}`. The `CONTENT` can be almost anything. Currently, syntax highlighting for more than 90 languages plus many dialects is available. For more details, especially how to change the syntax highlighting, see section 8.5.3.



The optional argument `OPTIONS` allows for passing additional settings to the current macro call. These settings are only applied locally and should be rarely necessary. The authors do not advise the usage of local options, which could lead to inconsistencies.



In alpha-2022-04-30, the usage of \LaTeX special characters (`\`, `$`, `&`, `{`, `}`, `%`, `~`) as `CONTENT` in `\daListingInline{CONTENT}` leads to compilation errors, especially if the macro is called in highlight boxes or tables. If the compilation halts and does not produce a valid pdf output, try using `\lstinline{CONTENT}`. The last resort is to use `\lstinline$CONTENT$` in combination with escaping the \LaTeX special characters.

8.5.2. Inblock

The main design goal of the docART Utility is to provide automatic import functionality for source code into the finished documentation and keeping the listed code up to date. In addition, it is possible to load either the complete file or only parts of it (which is helpful for writing tutorials) by a tag system.

The labeling system that the docART Utility uses is inserted as a special comment into the source code of the file that should be loaded. The structure of the docART listing comment is as follows:

```
COMMENT DELIMITER TAG DELIMITER
```

By default, the `DELIMITER` is set to `+++` (this can be changed by redefining `\daTagDelim` in `./settings/listingsSettings.tex`). Examples on how to construct valid labels for different languages and delimiters can be found in table 8.7.



docART Utility Handbook



language	comment	delimiter	source code
C++	//	+++	//+++TAG+++
		@@@	//@@@TAG@@@
		!	//!TAG!
Python	#	+++	#+++TAG+++
		@@@	#@@@TAG@@@
		!	#!TAG!
L ^A T _E X	%	+++	%+++TAG+++
		@@@	%@@@TAG@@@
		!	%!TAG!
Lua	--	+++	--+++TAG+++
		@@@	--@@@TAG@@@
		!	--!TAG!

Table 8.7: Examples for valid docART listing comments for different languages and delimiters.

Since the DELIMITER is part of a comment, any combination of ASCII characters is valid.



- The DELIMITER has to be the same for all the imported files.
- docART macros only need the TAG as input. The COMMENT as well as the leading/trailing DELIMITER may not be provided.

To illustrate how the docART listing comments work, consider one wants to write a tutorial about the Python script presented in listing 8.9.

```
# -*- coding: utf-8 -*-
#+++Step1And2+++
import os
#+++Step1+++
from datetime import datetime

directory = "./test-directory/"
#+++Step1+++

if not os.path.exists(directory):
    os.makedirs(directory)

#+++Step1+++
filename = datetime.now().strftime("%Y_%m_%d-%I_%M_%S_%p")
filepath = directory + filename
open(filepath, 'a').close()
#+++Step1+++
#+++Step1And2+++
```

Listing 8.9: An example for the import of a Python script as a code listing. The listing shows the usage of the docART listing comments in the source code.



The Python script shown in listing 8.9 contains two different docART listing comments (#+++Step1+++ and #+++Step1And2+++), with the first one being present four times, whereas the latter one is only seen twice. Calling

```
\daListingCaptionBelow{%  
  ./examples/code/docart_regex-labels_examples/docart_regex-labels_example1.py%  
}{Step1}{lst:bf:listing:inblock:python-regex-example1:step1}{%  
  Resulting listing for the tag \enquote{\lstinline$Step1$}. %  
}
```

with the TAG part (“Step1”) of the docART listing comment #+++Step1+++ leads to the code listing with caption shown in listing 8.10.

```
from datetime import datetime  
  
directory = "./test-directory/"  
filename = datetime.now().strftime("%Y_%m_%d-%I_%M_%S_%p")  
filepath = directory + filename  
open(filepath, 'a').close()
```

Listing 8.10: Resulting listing for the tag “Step1”.

The resulting listing includes all the code which was sandwiched between two consecutive occurrences of #+++Step1+-. If there are integer multiple of 2 occurrences of the label, the listing will only contain the code inside those occurrences; the rest of the file will be omitted. Calling

```
\daListingCaptionAbove{%  
  ./examples/code/docart_regex-labels_examples/docart_regex-labels_example1.py%  
}{Step1And2}{lst:bf:listing:inblock:python-regex-example1:step1And2}{%  
  Resulting listing for the tag \enquote{\lstinline$Step1And2$}. %  
}
```

with the TAG part (“Step1And2”) of the docART listing comment #+++Step1And2+++ leads to the code listing with caption above as shown in listing 8.11.

Listing 8.11: Resulting listing for the tag “Step1And2”.

```
import os  
from datetime import datetime  
  
directory = "./test-directory/"  
  
if not os.path.exists(directory):  
    os.makedirs(directory)  
  
filename = datetime.now().strftime("%Y_%m_%d-%I_%M_%S_%p")  
filepath = directory + filename  
open(filepath, 'a').close()
```



docART Utility Handbook



Comparing listing 8.10 with listing 8.11, the latter listing shows more file content. Also worth mentioning is that all the docART listing comments are not displayed in the listing. Besides using self-defined docART listing comments in the source code for partial loading, there is the option to process the complete file without needing to specify one. The listing 8.12 is generated by using “all” as a tag input. Like previously, only the pure source code with standard language comments (without docART listing comments) gets displayed.

```
# -*- coding: utf-8 -*-
import os
from datetime import datetime

directory = "./test-directory/"

if not os.path.exists(directory):
    os.makedirs(directory)

filename = datetime.now().strftime("%Y_%m_%d-%I_%M_%S_%p")
filepath = directory + filename
open(filepath, 'a').close()
```

Listing 8.12: Resulting listing when using the “all” tag.

Especially for writing tutorials it can be helpful to not only show different parts of a file, but also add additional information and formatting in the listing. A slight modification to the example makes this possible (see listing 8.13).

```
# -*- coding: utf-8 -*-
####Step1And2####
import os
####Step1####
from datetime import datetime

directory = "./test-directory/"

####comment Step1#### FUTURE CODE FOR DIR CHECKER
####newline Step1####
####Step1####
# Check if dir exists and if not create it
if not os.path.exists(directory):
    os.makedirs(directory)

####Step1####
filename = datetime.now().strftime("%Y_%m_%d-%I_%M_%S_%p")
####comment Step1#### Generate the file path without checking if dir exists
####comment Step1And2, all #### file path generation save (dir check above)
filepath = directory + filename

open(filepath, 'a').close()
####Step1####
####Step1And2####
```

Listing 8.13: An example for the import of a Python script as a code listing. This file is a slight variation of the one shown in listing 8.9.



docART Utility Handbook



Comparing the resulting code listings 8.14 to 8.16 reveals that comments and newlines (e.g. `####comment Step1####` `COMMENT` and `####newline Step1####`) get only displayed when the label passed to the listing macro is identical. If multiple labels are specified as a comma separated list (e.g. `####comment Step1And2, all ####`), the element is present in all the listings generated with matching labels. This is true for user-defined labels as well as “all”.

```
from datetime import datetime

directory = "./test-directory/"

# FUTURE CODE FOR DIR CHECKER

filename = datetime.now().strftime("%Y_%m_%d-%I_%M_%S_%p")
# Generate the file path without checking if dir exists
filepath = directory + filename

open(filepath, 'a').close()
```

Listing 8.14: Resulting listing for the modified Python script when using the tag “Step1”.

```
import os
from datetime import datetime

directory = "./test-directory/"

# Check if dir exists and if not create it
if not os.path.exists(directory):
    os.makedirs(directory)

filename = datetime.now().strftime("%Y_%m_%d-%I_%M_%S_%p")
# file path generation save (dir check above)
filepath = directory + filename

open(filepath, 'a').close()
```

Listing 8.15: Resulting listing for the modified Python script when using the tag “Step1And2”.

```
# -*- coding: utf-8 -*-
import os
from datetime import datetime

directory = "./test-directory/"

# Check if dir exists and if not create it
if not os.path.exists(directory):
    os.makedirs(directory)

filename = datetime.now().strftime("%Y_%m_%d-%I_%M_%S_%p")
# file path generation save (dir check above)
filepath = directory + filename

open(filepath, 'a').close()
```

Listing 8.16: Resulting listing for the modified Python script when using the tag “all”.



docART Utility Handbook



The authors recommend to always use listings with captions and labels (like generated by `\daListingCaptionBelow`, `\daListingCaptionAbove`), since this helps the reader to compare different steps due to referencing (see section 8.6). Sometimes the code listing is too long/-complicated to fit in the flow of the running text. For this scenario, the docART Utility provides the macro `\daListingCaptionOff{FILE}{TAG}`, which provides the same functionality as the macro variants described before, but does neither generate a caption nor can be referenced. This option is used throughout the docART Utility documentation to show case more complicated \LaTeX code snippets. An example would be on page 28, where the syntax for `\daListingCaptionBelow` is shown. As reference, this specific syntax example was generated by

```
\daListingCaptionOff[style=LaTeX]{%  
  ./document_parts/basic_functionality_parts/docart_documentation_bf_listings_  
  v0.tex%  
}{RegexLabelsExample1Step1}
```

For details about the optional argument `style=LaTeX`, please see section 8.5.3. Besides the syntax highlighting, the listing commands provide automatic line break (inline + inblock) as well as automatic page break (inblock) functionality. Please note that the user has no control over these features (except adapting the loaded source code file). For in depth examples how the line and page break behave, refer to the source code listings in appendix D.



8.5.3. Changing the Listing Syntax Highlighting



The descriptions provided in this section are only valid for version alpha-2022-04-30 and will definitely change in a future release.

By default, the docART Utility supports syntax highlighting for 95 languages plus 66 dialects. An overview of the available languages is provided in table 8.8, whereas the dialects are outlined in table 8.3.

To get a very basic syntax highlighting, one could use the macro

```
\lstset{language=[DIALECT]LANGUAGE}
```

to select the correct lexer. It is highly recommended to specify the dialect when possible to obtain the best syntax highlighting possible.



The authors do not advise to specify the language, but instead recommend using styles (see page 33). The reasons being:

- Without a style, the highlighting is limited to bold, italics, etc..
- The amount of detected keywords is very limited.
- Defining custom languages is tricky (custom styles easier).

ABAP	ACM	ACMscript	ACSL	Ada	Algol
Ant	Assembler	Awk	bash	Basic	C
C++	Caml	CIL	Clean	Cobol	Comal 80
command.com	Cmsol	csh	Delphi	Eiffel	Elan
elisp	erlang	Euphoria	Fortran	GAP	GCL
Gnuplot	Go	hansl	Haskell	HTML	IDL
inform	Java	JVMIS	ksh	Lingo	Lisp
LLVM	Logo	Lua	make	Mathematica	Matlab
Mercury	MetaPost	Miranda	Mizar	ML	Modula-2
MuPAD	NASTRAN	Oberon-2	OCL	Octave	OORexx
Oz	Pascal	Perl	PHP	PL/I	Plasm
PostScript	POV	Prolog	Promela	PSTricks	Python
R	Reduce	Rexx	RSL	Ruby	S
SAS	Scala	Scilab	sh	SHELXL	Simula
SPARQL	SQL	Swift	tcl	TeX	VBScript
Verilog	VHDL	VRML	XML	XSLT	

Table 8.8: Overview of all by the docART Utility supported languages for syntax highlighting.

Source: The Listings Package manual, 2020/03/24, Version 1.8d, p. 13, available at:

<https://ftp.tu-chemnitz.de/pub/tex/macros/latex/contrib/listings/listings.pdf>



language	dialects	default	language	dialects	default
ABAP	R/2 4.3, R/2 5.0, R/3 3.1, R/3 4.6C, R/3 6.10	R/3 6.10	Ada	83, 95, 2005	2005
Basic	Visual	—	Algol	60, 68	68
C++	11, ANSI, GNU, ISO, Visual	ISO	Assembler	Motorola68k, x86masm	—
command.com	WinXP	WinXP	Awk	gnu, POSIX	gnu
Fortran	77, 90, 95, 03, 08	95	C	ANSI, Handel, Objective, Sharp	ANSI
Lua	5.0, 5.1, 5.2, 5.3	—	Caml	light, Objective	light
Mathematica	1.0, 3.0, 5.2, 11.0	11.0	Cobol	1974, 1985, ibm	1985
Rexx	empty, VM/XA	—	IDL	empty, CORBA	—
S	empty, PLUS	—	Java	empty, AspectJ	—
tcl	empty, tk	—	Lisp	empty, Auto	—
TeX	ALLaTeX, common, LaTeX, plain, primitive	plain	make	empty, gnu	—
			OCL	decorative, OMG	OMG
			Pascal	Borland6, Standard, XSC	Stand- ard
			Simula	67, CII, DEC, IBM	67
			VHDL	empty, AMS	—
			VRML	97	97

Table 8.3: Overview of all by the docART Utility supported language dialects for syntax highlighting. Source: The Listings Package manual, 2020/03/24, Version 1.8d, p. 13, available at: <https://ftp.tu-chemnitz.de/pub/tex/macros/latex/contrib/listings/listings.pdf>

In comparison to obtain the syntax highlighting by setting the `language` option, using `style` has many advantages:

- `style` allows for complex highlighting schemes combining typefaces, fonts and color.
- Additional keywords with different styling options can be added (allows to customize the lists of available keywords very easily).

The major drawback is that the language, dialect and keywords are set within the style definition, so there is no separation between functionality and styling. For the user this could mean multiple styles for the same language as well as copy-pasting settings between them. The docART Utility provides as default four styles: C++, LaTeX, Lua and Python. They can be set by `\lstset{style=STYLE}` and are used for syntax highlighting until another style is set. The following listings 8.17 to 8.20 showcase a simple “Hello World”-example for all the default styles. For additional, more complicated examples, see appendix D.



docART Utility Handbook



```
// Requires C++ 17 or higher

#include <iostream>
#include <ctime>

int main() {
    std::time_t result = std::time(nullptr);
    std::cout << "Hello World, it is " << std::asctime(std::localtime(&result));
    return 0;
}
```

Listing 8.17: An “Hello World”-example written in C++ showcasing the default docART C++ syntax highlighting style.

```
\documentclass{scrreprt}

\usepackage[yyyymmdd,hmmss]{datetime}

\begin{document}
    Hello World, the date is \today, the time is \currenttime.
\end{document}
```

Listing 8.18: An “Hello World”-example written in \LaTeX showcasing the default docART LaTeX syntax highlighting style.

```
local dateString = os.date("%Y-%m-%d")
local timeString = os.date("%I:%M:%S %p")
local preString = "Hello World, the date is "
local ostring = preString .. dateString .. ", the time is " .. timeString .. "."
```

Listing 8.19: An “Hello World”-example written in Lua showcasing the default docART Lua syntax highlighting style.

```
# -*- coding: utf-8 -*-

from datetime import datetime

# generate the current date and time strings from time stamps
date = datetime.now().strftime("%Y-%m-%d")
time = datetime.now().strftime("%I:%M:%S %p")

# assemble the string which will be printed
# Warning: This syntax requires Python 3.6 or higher
ostring = f"Hello World, the date is {date}, the time is {time}."

print(ostring)
```

Listing 8.20: An “Hello World”-example written in Python showcasing the default docART Python syntax highlighting style.



8.5.4. Listings: Summary

The docART Utility, version alpha-2022-04-30, is built upon/provides wrapper macros for the listings package.

Available macros:

- Inline: (automatic line break)
 - `\daListingInline`[OPTIONS]{CODE}
 - Special cases (environments, etc.): `\lstinline`[OPTIONS]\$CODE\$ with escaping of \LaTeX special characters.
- Inblock: (automatic line + page break)
 - `\daListingCaptionOff`[OPTIONS]{FILE}{TAG}
 - `\daListingCaptionBelow`[OPTIONS]{FILE}{TAG}{LABEL}{CAPTION}
 - `\daListingCaptionAbove`[OPTIONS]{FILE}{TAG}{LABEL}{CAPTION}

List sections from a file with docART listing comments:

- docART listing comment syntax:
COMMENT DELIMITER TAG DELIMITER,
e.g. `####LABEL###` for Python with the default docART listing comment delimiter “+++”.
- docART listing comment options:
 - Comments:
e.g. `####comment LABEL1, ..., LABELN### # A comment`
 - Newlines:
e.g. `####newline LABEL###`
- Special tag: Passing “all” as TAG generates a listing of the complete file.
- All docART listing comments will be removed from the listing.

Syntax highlighting:

- Setting the language (lexer):
`\lstset{language=[DIALECT]LANGUAGE}`
- Setting a style (lexer + highlighting scheme + additional keywords):
`\lstset{style=STYLE}`
- Pre-made docART styles for C++, \LaTeX , Lua, Python.
- Additional keywords can be defined in the style definition.



8.6. References and Labels

8.6.1. Usage

docART and \LaTeX in general provide a very simple interface for setting labels and referencing them. The interface works the same for referring to chapters, sections, etc. or objects like figures, tables, code listings, etc.. In section 8.6.1, on page 36, the basic functionality will be explained.



It requires up-to two compilations until changes in references and labels take effect.

To set a label in any (text) position of the document, one uses the `\label{LABEL}` command with LABEL being an unique identifier which the user can choose (almost) freely. Referencing can be done via `\ref{LABEL}` or `\pageref{LABEL}`, which return the current table-of-content layer or the page the LABEL is placed. The two references at the beginning of this page were generated by the code shown in listing 8.21.

```
\subsection{Usage}
\label{subsec:bf:labels:usage}
\productName~and \LaTeX~in general provide a very simple interface for
setting labels and referencing them. The interface works the same for
referring to chapters, sections, etc. or objects like figures, tables, code
listings, etc.. In \mbox{section \ref{subsec:bf:labels:usage}}, on
\mbox{page \pageref{subsec:bf:labels:usage}}, the basic functionality will
be explained.
```

Listing 8.21: An example how to refer to certain text passages of the document.

For docART objects like figures, tables and code listings, when you choose a command variant which provides a caption, a label is automatically generated. As an example, listing 8.21 is generated by the code below.

```
\daListingCaptionBelow{%
./document_parts/basic_functionality_parts/docart_documentation_bf_labels_v
0.tex%
}{refLabelTextExample}{lst:bf:labels:usage:example-labels-in-text}{%
An example how to refer to certain text passages of the document.%
}
```

For this example, the label `lst:bf:labels:usage:example-labels-in-text` is automatically created and can be referred to like described above. The only difference is that `\ref{LABEL}` will return the number of the object instead of the table-of-content layer. For an overview what `\ref{LABEL}` returns depending upon the object, please refer to page 37, table 8.4.



8.6.2. Recommended Label Naming Convention

Although label names may contain any character, it is recommended to waive using \LaTeX special or non-standard-ASCII characters like umlaute.

A suggestion from the \LaTeX community is to use label names that contain the type, followed by a colon and a brief description of the object that is labeled:

TYPE:DESCRIPTOR

The list of recommended TYPE identifiers is given in table 8.4.

object	\LaTeX type	return type	TYPE-id
chapter	text	toc-layers	chap
section	text	toc-layers	sec
subsection	text	toc-layers	subsec
subsubsection	text	toc-layers	subsubsec
figure	float	object number	fig
table	float/longtable	object number	tab
listing	verbatim	object number	lst

Table 8.4: Recommended TYPE identifiers for different type of labeled objects.

For example, if one wants to refer to table 8.4 which contains the recommended Type identifiers, a possible label would be:

tab:recommended-type-ids

Please note that instead of white spaces, dashes were used in the descriptor to eliminate any chance of parsing problems in the \LaTeX -Backend.

The authors of the docART Utility recommend a variant of this scheme. Especially for documentations, when multiple individuals have to work on different parts of the document, it is very helpful to provide further information encoded within the label name; especially where the label is located. For the docART Utility documentation, the following scheme has been used:

TYPE:TOLAYERS:DESCRIPTOR

For example, the label for table 8.4 looks like this:

tab:bf:labels:naming-convention:recommended-type-ids

The label consists of the TYPE identifier, which is followed by colon-separated toc-layers (in this case: abbreviated names of the respective chapter, section, subsection) before being ended by the DESCRIPTOR. This makes the label name a lot longer and leads to issues when toc-element names change. However, the authors' experience shows that if the outline of the document is planned before hand (what always should happen for any good documentation), this approach facilitates collaborative writing.



9. Adapting the Default docART Theme

One of the design goals of the docART Utility was to provide a tool that is almost ready to use and only needs minimal setup. If one does not want to change the colors, page margin/type area or fonts, all that has to be done is make the changes outlined in section 9.1, section 9.2, section 9.3.1 and section 9.3.3.

9.1. Project Details

For changing the project details and pdf metadata information, navigate to the `./settings/` folder. The file `projectDetails.tex` contains the following macros:

macro	description
<code>\productName</code>	A macro storing the product name
<code>\releaseState</code>	A macro storing the release status of the product (alpha, beta, release, ...)
<code>\releaseDate</code>	A macro storing the release date of the product
<code>\productVersion</code>	A macro storing the product version default: <code>\releaseState-\releaseDate</code>
<code>\documentRevision</code>	A macro storing the number of documentation revisions
<code>\docRevText</code>	A macro storing a version of the number of documentation revisions with a string prefix default: Doc. Rev. <code>\documentRevision</code>



The macros defined in `projectDetails.tex` are not mandatory for the functionality of the docART Utility. They are designed to facilitate the usage of important, repeating expressions and to ensure consistency. If one decides not to use them, special care has to be taken during the configurations outlined in section 9.3.1 and section 9.3.3; otherwise the document may not compile.

The authors highly encourage the users of the docART Utility to use these macros and create new ones as needed. New project dependent macros should be placed in `./settings/projectDetails.tex`, since this is one of the first settings files to be loaded at compile time. This allows other files and macros to inherit the changes.



9.2. pdf Metadata

Although not strictly necessary, the authors highly recommend setting pdf metadata information. This not only helps the (pdf) reader, but also the documentation author(s) to keep track of changes without solely relying upon file names.

metadata tag	description
pdftitle	default: <code>\productName~Documentation</code>
pdfsubject	default: Documentation for <code>\productName</code> , <code>\productVersion</code> , <code>\docRevText</code>
pdfauthor	default: <code>\productName~Utility</code>
pdfkeywords	default: Documentation, coding, <code>\LaTeX</code> , <code>\productName</code>
pdfcreator	default: docART Utility, alpha-2022-04-30; Backend: (Lua)\LaTeX~with hyperref



If one decided neither to use project detail macros (see section 9.1) nor pdf metadata, one has to delete the contents of the file `./settings/pdfMetadata.tex`, but not the file itself!



Custom pdf metadata keywords are not possible with docART, alpha-2022-04-30, and probably with plain (Lua)LaTeX without post-processing in general. The other options `\pdfinfo` or loading a XMP-file via the `hyperxmp-usepackage` provide some additional keywords, but also lack support for custom entries.



9.3. Page Setup

9.3.1. Title Page

The title page generated by the docART Utility is defined within the `\daGenerateTitlepage` macro. The user can change the title page content by modifying the macro definitions in `./settings/titlepageContent.tex`. The available macros with a short description are listed below.

macro	postion	recommended
<code>\daTitlepageTitle</code>	top most entry	document title
<code>\daTitlepageLogo</code>	below title	product/company logo
<code>\daTitlepageDescription</code>	below logo	short product description
<code>\daTitlepageAuthor</code>	below description	author of document
<code>\daTitlepageDocumentInformation</code>	below author	type of document/version
<code>\daTitlepageLicense</code>	entry at bottom	licensing information

If e. g. a different layout is required, the `\daGenerateTitlepage` macro definition can be changed by editing the file `./docart-utility/class-master-theme/titlepage.dcmnts`, which is shown in listing 9.1.

```
\newcommand{\daGenerateTitlepage}{  
  \begin{titlepage}  
    % TITLE PAGE CONTENT  
  \end{titlepage}  
}
```

Listing 9.1: Content of the file `./docart-utility/class-master-theme/titlepage.dcmnts`.

The TITLE PAGE CONTENT can be virtually anything. However, for most use cases, simply adapting the example file by replacing the content should be enough.



Making more substantial changes to the title page requires basic knowledge of \LaTeX . In the experience of the authors, a web search for “LaTeX title page” will lead to easily adaptable results. A good source is the Overleaf guide “How to Write a Thesis in LaTeX”, part 5.



9.3.2. Margins and Type Area



In version alpha-2022-04-30, the page setup is hard coded in the docART \LaTeX class file. If one has some \LaTeX experience, it can be changed by manipulating the geometry-package options.

9.3.3. Page Header and Footer

The page header/footer are defined in `./docart-utility/class-master-theme/header.dcm`ts and `./docart-utility/class-master-theme/footer.dcm`ts respectively. The table 9.4 gives an overview of the options provided by the `scr`layer-`scr`page package, on which the docART header/footer are built upon.

macro	sets the ...	location
<code>\ihead{CONTENT}</code>	inner header	top left
<code>\chead{CONTENT}</code>	center header	top center
<code>\ohead{CONTENT}</code>	outer header	top right
<code>\ifoot{CONTENT}</code>	inner footer	bottom left
<code>\cfoot{CONTENT}</code>	center footer	bottom center
<code>\ofoot{CONTENT}</code>	outer footer	bottom right

Table 9.4: Overview of page header and footer options (location only valid for single page layouts).

The content of any header or footer element can be as complex as one wishes. A possibility is to repeat the section title in the center head and have the page number as center foot. This can be achieved by changing `./docart-utility/class-master-theme/header.dcm`ts to

```
\automark[section]{chapter}  
\chead{\leftmark}
```

and `./docart-utility/class-master-theme/footer.dcm`ts to

```
\cfoot[\pagemark]{\pagemark}
```



The content in header and footer does not have automatic line break functionality. If the header/footer are comprised of multiple elements and long chapter/section titles are used, overlaps of content can occur.



For more control over the alignment of header/footer elements, the authors recommend the usage of TikZ. However, this requires experience with \LaTeX and familiarity with the basics of TikZ.



9.4. Fonts

The fonts can be changed in `./docart-utility/class-master-theme/fontSettings.dcm`.

```
\setmainfont[%  
  Path=./docart-utility/fonts/,%  
  BoldFont={texgyreheros-bold.otf},%  
  ItalicFont={texgyreheros-italic.otf},%  
{texgyreheros-regular.otf}  
  
\setmonofont[%  
  Path=./docart-utility/fonts/,%  
  BoldFont={Hack-Bold.ttf},%  
  ItalicFont={Hack-Italic.ttf},%  
{Hack-Regular.ttf}
```

Listing 9.2: Overview of the default docART Utility font settings located at `./docart-utility/class-master-theme/fontSettings.dcm`.

The docART Utility supports the `otf`- as well as the `ttf`-font-format. It is required to provide the path to the font file (in this case `./docart-utility/fonts/`) as well as the file names for normal, bold and italic typefaces like shown in listing 9.2.



In version alpha-2022-04-30, the way the docART \LaTeX class is written only supports `otf/ttf`-fonts that have separate files for normal, bold and italic typefaces. Additionally, the default font size is fixed in the docART class file to 12 pt.



Despite being possible, the authors of the docART Utility do not recommend to use system fonts. First of all, if collaborators are using different (versions of) operating systems with different software installed, the list of available fonts will not be consistent. Secondly, some system fonts do not cause loading errors and consequently preventing a successful compilation.

For the main text font (set by `\setmainfont`), a sans-serif type is recommended, whereas for the code snippets a monospace font is the best choice. By default, the docART Utility uses “TeX Gyre Heros” as main text font and “Hack” for code snippets/listings.

font	example
“TeX Gyre Heros”:	The quick brown fox jumps over the lazy dog. Aa Bb Cc Ii Jj Ll Oo 0123456789
“Hack”:	The quick brown fox jumps over the lazy dog. Aa Bb Cc Ii Jj Ll Oo 0123456789



9.5. Colors

The color settings are split into two files: The file `./settings/colorSettings.tex` is currently intended for changes by the end user (like adding custom colors or changing the listing color scheme). The file `./docart-utility/class-master-theme/default-colors.dcms` is primarily intended for docART theme maintainer, since it determines the fundamental color scheme of the generated document.



The split of the settings and the following explanations are only correct for alpha-2022-04-30 and will change in future releases.

New colors can be defined by `\definecolor{NAME}{MODEL}{VALUE}`. The authors recommend to use `rgb` or `HTML` as color models. Creating a name variant of an already existing color is achieved by `\colorlet{NEW COLOR}{EXISTING COLOR}`.

Changing the default docART color scheme:

- To change the existing code snippet color scheme, adapt the color definitions of `daListingColorLineNumbers`, `daListingColorComment`, `daListingColorBackground` and `daListingColorString` in `./settings/colorSettings.tex`.
- To modify the table highlighting color, change the definition of `tableRowHighlightColor` in `./docart-utility/class-master-theme/default-colors.dcms`.
- Adapting the color of headlines and table-of-content entries, modify the value of `headlinecolor` in `./docart-utility/class-master-theme/default-colors.dcms`.



10. Known Bugs

1. Some pdf viewers show white lines between the rows in a code listing. This also depends on the zoom level.
2. \LaTeX listings have issues that all words in text identical to keywords will be highlighted; cannot use `\` for regex/identification.
3. Highlight boxes can currently not be made from a template and have to be coded from scratch.
4. At least two, sometimes three compilations required until tables have correct layout/optics.
5. Table row highlighting sometimes dependent upon tables that came before.
6. Icon alignment is not accurate, especially in tables.
7. `\daListingInline` and `\lstinline` mess up spacing over line break.
8. `\daListingInline` requires escaping of \LaTeX special characters passed in as an argument.
9. Indentation correction for listings randomly does not work properly.
10. Last line in listings sometimes removed. Seems to be related with the usage of the “all” tag. Fix: Add empty line to file.
11. Empty lines in Lua scripts will be removed.
12. Setting any expression as `emph` in any listing style will affect all other styles.
13. Two sections on the same page have a way too big vertical blank space in between. Fix: `\vspace{-1.5cm}` before the second `\section` call.
14. No proper math font loaded. The default \LaTeX math font does not fit the rest of the fonts (size, shape).
15. The aux-files on rare occasion have to be manually deleted before the compilation (otherwise compilation will not finish successfully). Observed case: Changing command name of macro which was present in a \LaTeX list (table of contents, section argument, etc.). Old command name still present in aux-file, which led to an aborted compilation before it could be completed (aux-files therefore did not get properly recreated).



Appendix



A. Contributors

Martin Stimpfl: original idea, prototyping, design requirements, architecture, Python scripting, hashing and regex expert, examples, debugging, proof-reading documentation

Daniel Sacré: prototyping, design requirements, architecture, Lua scripting, \LaTeX expert, examples, main author documentation



B. Licensing

B.1. Mandatory Products

B.1.1. \LaTeX -Backend

There are two options for the \LaTeX -Backend: Either `MikTeX` or `texlive`. Both of them are a collection of sub-modules, which all have their own license, e.g. MIT, GNU Public License (GPL), LaTeX Project Public License (LPPL) and others. Currently the consent is that both \LaTeX -Backends are redistributable according to the Free Software Foundation's (FSF) definition and the Debian Free Software Guidelines under the constraint that one satisfies all the requirements placed by the owners of the respective bundled packages.

This has no effect for the usage of the docART Utility as long as one is not planning to ship any part of the \LaTeX -Backend with the version of ones product.

Option 1: `MikTeX`

These links lead to the websites that elaborate on the licensing situation of `MikTeX`.

<https://miktex.org/copying>

<https://www.gnu.org/philosophy/free-sw.html>

<https://www.debian.org/intro/free>

Option 2: `texlive`

The situation for `texlive` is very similar to `MikTeX`. Information can be found on the website of the \TeX Users Group (TUG):

<https://tug.org/texlive/LICENSE.TL>



B.1.2. Python

Similar to the situation with the \LaTeX -Backend, the Python Standard Library (PSL) is a collection of modules, each of which has its own specific license. In contrast to the \LaTeX -Backend, there is a dedicated license for redistribution of the PSL: The Python Software Foundation (PSF) issued the PSF License Agreement, which is currently compatible to the GNU Public License (GPL). For further reading, visit the following websites:

<https://docs.python.org/3/license.html#psf-license>

<https://www.gnu.org/licenses/gpl-3.0.en.html>

Like mentioned previously for the \LaTeX -Backend, the licensing situation has no effect for the usage of the docART Utility as long as one is not planning to ship any part of the PSL with the version of ones product.

For context, the docART depends upon the following Python modules:

module	PSL	license
os	yes	PSF
sys	yes	PSF
re	yes	PSF
time	yes	PSF
hashlib	yes	PSF
xml.etree.ElementTree	yes	PSF
csv	yes	PSF

B.1.3. Lua Hash Library

docART uses the Lua Hash Library VERSION: 9 (2020-05-10) written by Egor Skriptunoff, released under the MIT License.

Links:

https://github.com/Egor-Skriptunoff/pure_lua_SHA

https://github.com/Egor-Skriptunoff/pure_lua_SHA/blob/master/LICENSE

<https://opensource.org/licenses/MIT>



B.1.4. docART Utility

Almost all the files (exceptions: see warning boxes below) provided within the docART Utility are licensed under GNU General Public License (GPLv3). This includes all \LaTeX -, Python-, and Lua-files as well as the docART specific dcmts-files and the documentation (pdf as well as source).

If you are re-using some part of the docART Utility, please give credit according to the GPLv3 rules as follows:

Martin Stimpfl, Daniel Sacré: docART Utility, alpha-2022-04-30,
License: GNU General Public License (GPLv3).



The docART logo is the only exemption: It is released under Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0). This means reusing the logo partially or completely requires giving credit:

Martin Stimpfl, Daniel Sacré: docART logo, alpha-2022-04-30,
CC BY-SA 4.0

Please respect the CC clause that re-usage of logos, trademarks, etc. should be done in a way that does not suggest that the original copyright holder endorses the person who re-uses the object.

Links:

<https://github.com/d-sacre/docart-documentation-utility/blob/current-release/LICENSE>
repository: [/docart-utility/class-master-theme/cmt-images/license_cc-by-sa-4-0.txt](#)
<https://www.gnu.org/licenses/gpl-3.0.en.html>
<https://creativecommons.org/licenses/by-sa/4.0/deed.en>



When to give credit, re-publish the source/pdf and under which license?

Q: I wrote a documentation with the docART Utility for a customer.

Do I have to publish my (L^AT_EX) source code and under which license?

Do I have to publish the pdf under the GNU General Public License (GPLv3)?

Do I have to give credit to (the authors of) the docART Utility?

A: **No.** The (L^AT_EX) source code does not have to be published. Like described on the last page, most of the content of the L^AT_EX file is data specified by you, so either you or your customer own the rights (depending on the agreement you have with your customer). However, the authors recommend to provide the (L^AT_EX) source code to the customer so that the customer can do changes on his/her/its own.

The pdf may be distributed under any terms that you or your customer specifies. It does not have to be made publicly available.

It is not necessary to give credit to the docART Utility. However, if it is possible for you to give a little shout-out (something like "This documentation has been made with the docART Utility.") and perhaps even provide a link to the github repository, this would help to find new contributors and improve the product.

Q: I designed my own theme or made changes to the default theme (dctms-files) and files under ./settings/.

Do I have to publish my (L^AT_EX) source code and under which license?

A: **Depends.** There are two cases:

Case 1: Changes to setting files and dctms-files without license header: **No.**

The authors consider changes in these files as user settings, which do not fall under copyright. If you think others could benefit from your theme, the authors encourage you to publish it. Despite there is no legal obligation, the authors would like to ask the publisher to use a permissive OpenSource license (ideally GNU General Public License (GPLv3) to keep it consistent with the docART Utility) and give a shout-out to the docART Utility.

Case 2: Changes to dctms-files with license header: **Sort of.**

The dctms-files with license header contain macro or environment definitions that are the back bone of the docART Utility. As long as these macros do not get changed, the rest of the file content has to be seen as user settings and therefore does not fall under copyright. If these themes are just used e.g. in a company internally, a publication is not required. As soon as this theme gets bundled with a product that gets sold, the full extend of GNU General Public License (GPLv3) is valid (mandatory publication under same license).

This is the status for version alpha-2022-04-30. The authors of the docART Utility plan to separate user settings and protected code more vigilantly in one of the next releases.



docART Utility Handbook



- Q:** I did some changes to the original docART Utility and want to sell it as a product or offer (payed) services with my altered version.
Do I have to publish my (\LaTeX) source code and under which license?
- A:** **Yes.** This situation is considered a fork, which is allowed by the GNU General Public License (GPLv3) as long as the altered source code is published with the same license or any later version (at your discretion) and the reference to the original product. Additionally, your product has to be published under a different name and may not use any part of the original docART Utility (documentation, logos, etc.) outside the cases specified in the respective license agreements (especially regarding advertising purposes).
- Q:** I wrote a wrapper macro for an already existing docART command.
Do I have to publish my source code and under which license?
- A:** **No.** This change does not add new functionality to the docART Utility and has more to do with quality-of-life improvements. If you think your wrapper macro is helpful and others could benefit from it, the authors encourage you to publish it. If you publish it, the license has to be GNU General Public License (GPLv3) or any later version (at your discretion), including a reference to the original docART Utility release.
- Q:** I wrote a macro to extend the functionality of the docART Utility to suit my needs. It is neither based upon any in the docART Utility already existing elements nor does it not use any of the docART Utility's source code nor gets called by any element of it (docart-utility/macros).
Do I have to publish my source code and under which license?
- A:** **No.** In this case, you are not building upon the docART Utility like e.g. by writing a wrapper macro, but instead creating your own product. The \LaTeX philosophy is that the Backend only provides the general functionality and the end users write their own specific macros. Since the docART Utility is in itself just a user defined macro collecting that is published in the hope to be useful for somebody, the authors highly encourage the end user to extend the docART Utility's scope by their own ideas. The authors would love to see the extensions made publicly available (ideally as OpenSource under permissive licenses like GPL or MIT). If you do so, please respect the guidelines regarding re-usage of the docART Utility name, logo, etc. and if possible give a shout-out to the original project.
- Q:** I searched through the \LaTeX code of the handbook and found one neat code snippet that I want to re-use.
Do I need to give credit and have to publish my own product under GNU General Public License (GPLv3)?



docART Utility Handbook



A: Depends. The authors see the docART Utility Handbook as a knowledge base, which should be free to use to everyone for every case. If you found one neat \LaTeX trick or hack that you want to adapt for yourself, even outside of the scope of docART Utility, go for it; no need for publication under GNU General Public License (GPLv3) or giving credit (although the authors would be happy about a little shout-out if it is possible for you to do so). However, anything more like re-publishing substantial parts of the docART Utility Handbook has to be done under GNU General Public License (GPLv3) and giving credit appropriately.

Q: I want to re-publish (a part of) the docART Utility Handbook with some of my own changes.

Do I need to give credit and have to publish my own product under GNU General Public License (GPLv3)?

A: Yes. Always. This measure is required to ensure that one version of the docART Utility is always available for everyone. If a company internally uses the docART Utility with their own closed source extensions and want to write an updated handbook for internal use only, there can an exception be made. However, as soon as the company sells a service based upon this modified version, the full extend of GNU General Public License (GPLv3) applies.



B.2. Optional Products

B.2.1. Fonts

The docART Utility ships with two fonts (text and code listings). The fonts have been chosen such that their licenses allow any form of redistribution with except of font bundles. If the licenses of these fonts do not align with your needs, you can easily swap them out with others (see section 9).

TeX Gyre Heros

The “TeX Gyre Heros”-font is the main text font. It comes with a very liberal license; only requirement is that derivative works have to be released under a different name.

Links:

<https://tug.org/fonts/licenses/GUST-FONT-LICENSE.txt>

<https://www.latex-project.org/lppl/>

Hack

The “Hack”-font is used for code listings. It is released under the MIT and BITSTREAM VERA LICENSE. The latter one is more restrictive than the license of the “TeX Gyre Heros”-font. The BITSTREAM VERA LICENSE e. g. prohibits redistribution in a font bundle. The way the docART Utility uses the font should be uncritical.

Links:

<https://github.com/source-foundry/Hack/blob/master/LICENSE.md>

<https://www.fontsquirrel.com/license/Bitstream-Vera-Sans>

<https://opensource.org/licenses/MIT>

B.2.2. TeXstudio

TeXstudio is a \LaTeX IDE, which is especially helpful for beginners. Installing this software is not necessary for the usage of the docART Utility; however it makes it simpler. Its permissive GNU General Public License (GPLv2) makes it ideal for future docART software bundles.

Links:

<https://www.texstudio.org/>

<https://www.gnu.org/licenses/old-licenses/gpl-2.0.html>



C. Installation

C.1. Mandatory Steps

In this section, the installation procedure for the mandatory utilities/software packages will be outlined. First in general, than in more details for the respective operating systems (Windows: section C.1.1, Linux: section C.1.2, Mac OS: section C.1.3) and finally the installation of the docART utility itself.

Step 1: Check Compliance with IT Security Policy



The docART Utility uses shell-escape to provide its functionality. Please check BEFORE installing the utility, whether your IT security policy allows this.

Step 2: Install a pdf Reader

Although most operating systems provide a pdf Reader out-of-the-box, it is especially under Windows a good practice to install a reader which is known to work well with \LaTeX (e. g. see warning box below). In theory, this step could be done at a later time. However, doing it now has some advantages, especially if one is planing to install the optional dependency TeXstudio later on (see section C.2.1).

Recommendation:

Windows:	Sumatra PDF, https://www.sumatrapdfreader.org/free-pdf-reader , License: GNU General Public License (GPLv3)
Linux:	Evince, best via packet manager, License: GNU Public License (GPL-2.0-or-later)



Any Adobe Acrobat Pdf Reader is not ideal for use with docART and \LaTeX in general. When the pdf is opened with the Adobe products, it gets locked, which in turn means that a simultaneous re-compilation with \LaTeX is not possible. In short: For every compilation, the pdf in the Adobe products has to be closed and afterwards re-opened.



Step 3: Install Python 3

The docART utility requires Python 3 to run. If it is already installed on the machine, one does not need to do anything. If it is not installed, please follow the recommendations.

Recommendation:

Windows:	direct download from Python Software Foundation, https://www.python.org/
Linux:	If Python 3 is not already included in one's Linux base installation, install it via the packet manager.
Mac OS:	Currently requires Homebrew for Python 3 installation. See section C.1.3.



- Using Anaconda as a Python backend is not recommended. Python packages included in Anaconda can be different/older than in the official Python Software Foundation/Community Releases. This might lead to docART not working properly. In addition, installing Anaconda besides the default Python installation on a Linux system can lead to conflicts.
- When having multiple Python installations available (especially on Linux and Mac OS or when using Anaconda), please check what the default Python version is that gets called via the python command. For docART alpha-2022-04-30, if python does not call Python 3.x, the docART utility will not work and crash.

Step 4: Install the \LaTeX -Backend

Both \LaTeX -Backends `texlive` and `MikTeX` are available for Windows, Linux and Mac OS. The choice is therefore mostly preference based.

Recommendation:

Windows:	MikTeX, direct download from: https://miktex.org/download
Linux:	texlive via the packet manager.
Mac OS:	MacTeX, direct download from: https://www.tug.org/mactex/mactex-download.html



- For some Windows (Enterprise) and `texlive` versions, the authors encountered a corrupted \LaTeX -Backend installation.
- Installing MikTeX under Linux can lead to font detection problems. Installation of \LaTeX packages from the Linux repositories is also not possible.
- For the \LaTeX -Backend installation to be recognized, it has to be completed before installing any optional software like TeXstudio.



Step 5: Install the Optional Components

Install all the required optional components. Especially for beginner or intermediate \LaTeX users, it is advantageous to install a \LaTeX IDE like Texmaker or TeXstudio (the latter one is recommended by the authors and its configuration/usage in conjunction with the docART utility will be outlined in section C.2.1, section 6.2 and section 7).

Step 6: Download and Unpack the docART Utility

Step 6.1: Download the zip-file or clone the official docART github repository:
<https://github.com/d-sacre/docart-documentation-utility>

Step 6.2: Only if you downloaded a zip-file: Unpack it into the folder you want to work in.

Step 7: Configuration

Before the first usage of the docART, the configuration outlined in section 6 has to be done.

C.1.1. Windows

A detailed step-by-step guide will be added in the next revision of the document.

C.1.2. Linux

A detailed step-by-step guide will be added in the next revision of the document.

C.1.3. Mac OS

In theory, the docART utility should work on any recent installation of Mac OS. However, due to not having access to a system running Mac OS, the authors of the utility could neither verify this nor experiment with the installation process.



At the time of writing this document, Mac OS still ships with Python 2.7 as default installation. Installing Python 3 can be achieved with Super User-level access via Homebrew. Sometimes Xcode has to be re-installed. To download the most up-to-date version of Xcode, a valid Apple ID is required.



C.2. Optional

C.2.1. TeXstudio

The TeXstudio IDE is a very useful tool since it combines a text editor and pdf viewer within one window, which provides immediate feedback and simplifies debugging. Additionally, it provides direct access to all \LaTeX functionality, has helper tools for automated code generation (very helpful for \LaTeX beginners) and automatically deals with complex compilation pipelines without the need of user input. Furthermore, the included pdf viewer allows compilation whilst the pdf is still open; this is especially handy if Adobe products are used as a default pdf viewer (see section C.1, p. 54).

Recommendation:

Windows:	direct download from: https://www.texstudio.org/
Linux:	texstudio via the packet manager.
Mac OS:	direct download from: https://www.texstudio.org/

Installation under Windows

- Step 1:** Open <https://www.texstudio.org/> in a web browser and scroll down/use the link to the “Download” section.
- Step 2:** Chose the correct TeXstudio version for your operating system (recommendation: Qt5 for stability; the new Qt6 should only be required for extremely high resolution monitors).
- Step 3:** After the download has finished, execute the installer. Administrator privileges might be required.



- Precompiled Windows Installer of the most recent TeXstudio version are only available for Windows 10, 64 bit. Older versions with support for different versions of Windows must be compiled from source.
- The portable version of TeXstudio might not have all the permissions that are required to run the docART utility.



Installation under Linux

The installation of TeXstudio under Linux should always be done via the packet manager. There are also Linux Appimages available, which should in theory work on any flavor of Linux.

Please note that for the installation one needs sudo permissions or has to be root.

Installation command:

Debian:	<code>sudo apt install texstudio</code>
Ubuntu:	<code>sudo apt install texstudio</code>
Arch Linux:	<code>sudo pacman -S texstudio</code>
OpenSUSE:	<code>zypper install texstudio</code>

Installation under Mac OS

- Step 1:** Open <https://www.texstudio.org/> in a web browser and scroll down/use the link to the “Download” section.
- Step 2:** Chose the correct TeXstudio version for your operating system (recommendation: Qt5 for stability; the new Qt6 should only be required for extremely high resolution monitors).
- Step 3:** After the download has finished, double click on the package file. It will unpack the disk image.
- Step 4:** When the disk image is unpacked, double click on it to start TeXstudio.



The TeXstudio developers “do not have an Apple Developer Account, so OS X may complain about an unidentified developer and deny opening” TeXstudio. Normally, there should be a pop-up asking one if one wants to execute software downloaded from the internet. If the pop-up does not appear or TeXstudio will even after confirmation not work, “open the context menu on the TeXstudio icon (Ctrl + Click) and select open”.

parts in quotes from: Remark on TeXstudio website besides Mac OS image in Download section.



D. Listing of Source Code

D.1. docART Utility

D.1.1. \LaTeX Class

Listing D.1: Contents of the docART \LaTeX class file.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{docart-utility/docart}[docart LaTeX class]

% Loading the scr-komascript base class
\LoadClass[12pt, a4paper, oneside,parskip=half,no indent]{scrreprt}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Packages %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Check if Settings File Exists and Load it
\RequirePackage{etoolbox}
\RequirePackage{xifthen}
\newcommand{\daCheckAndLoadSetting}[1]{
  \IfFileExists{#1}
  {\input{#1}}
  {
    \ClassError{docart}%
    {File #1 not found. An essential file could not be loaded}%
    {Please check if all required files are located in the correct position\
    MessageBreak according to the docart documentation.}%
  }
}

% Loading the project details at the begining, so that all the defined macros
  can be used
% by other dcmts/user setting files
\daCheckAndLoadSetting{settings/projectDetails.tex}

% Syntax Highlighting
\RequirePackage[procnames]{listings} %% alt: listings, pygment, listingsutf8
\RequirePackage{shellesc} % to be able to execute Python scripts via shell
\RequirePackage{luacode} % to be able to use Lua code directly within LuaLaTeX

\daCheckAndLoadSetting{settings/listingsSettings.tex}
\daCheckAndLoadSetting{./docart-utility/class-master-theme/listing.dcmnts}

%% Including Pictures
\RequirePackage{graphicx}

%% Colors
```



docART Utility Handbook



```
\RequirePackage[table,dvipsnames]{xcolor}
\daCheckAndLoadSetting{docart-utility/class-master-theme/default-colors.dcmds}
\daCheckAndLoadSetting{settings/colorSettings.tex}

\RequirePackage{tikz}
\usetikzlibrary{calc,positioning,decorations.text}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%% Formating the page, general typsetting options %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Fonts
\RequirePackage{fontspec}
\daCheckAndLoadSetting{docart-utility/class-master-theme/fontSettings.dcmds}

%% page margins and typearea
\RequirePackage[top=1.25cm,bottom=1.25cm,left=2cm,right=2cm,headheight=2.5cm,
includeheadfoot]{geometry}

\RequirePackage[automark]{scrlayer-scrpage}
\daCheckAndLoadSetting{docart-utility/class-master-theme/header.dcmds}
\daCheckAndLoadSetting{docart-utility/class-master-theme/footer.dcmds}

\pagestyle{scrheadings}
\renewcommand*{\chapterpagestyle}{scrheadings}

%% Settings for distance chapter to header
\renewcommand*{\chapterheadstartvskip}{\vspace*{-0.5cm}}
\renewcommand*{\chapterheadendvskip}{\vspace*{0.425cm}}

%% Settings for distance section and subsection
\RedeclareSectionCommand[beforeskip=-12.5ex,afterskip=2ex plus -.2ex]{section}

%% Formating the headlines
\setkomafont{disposition}{\color{headlinecolor}\bfseries}

%% Formating the captions
\RequirePackage[labelfont={bf}]{caption}
\captionsetup{format=plain,labelformat=simple,font=small,labelsep = colon}

%% Control over Bulletpoint Lists
\RequirePackage{enumitem}
\renewcommand\labelitemi{\textbf{\textbullet}}
\renewcommand\labelitemii{\textbullet}
\renewcommand\labelitemiii{\textbullet}

%% Titlepage
\daCheckAndLoadSetting{settings/titlepageContent.tex}
\daCheckAndLoadSetting{docart-utility/class-master-theme/titlepage.dcmds}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```



docART Utility Handbook



```
%% Highlight boxes
\daCheckAndLoadSetting{docart-utility/class-master-theme/highlightboxes.dcmnts}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Tables %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\RequirePackage{booktabs}
\RequirePackage{tabularx}
\RequirePackage{colortbl} % color in tables
\RequirePackage{multirow}

\RequirePackage{pgfplotstable}
\pgfplotsset{compat=1.17}

% %% Multipage Tables
\RequirePackage{longtable,tabu} % using tabu tables did clash with longtable/
    tabularx
% tabu at the moment just for table separator line command used

\daCheckAndLoadSetting{docart-utility/class-master-theme/tables.dcmnts}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Pictures %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\RequirePackage{graphicx}
\RequirePackage[]{}{subcaption}
\RequirePackage{xspace}

\daCheckAndLoadSetting{docart-utility/class-master-theme/figures.dcmnts}

\RequirePackage{tikz}[bclogo] % only used for some icons
\RequirePackage{ccicons} % provides Creative Commons License Icons

\daCheckAndLoadSetting{docart-utility/class-master-theme/icons.dcmnts}

%\daCheckAndLoadSetting{settings/pictureSettings.tex} % currently not needed;
    but in a later release
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\RequirePackage{csquotes}
\RequirePackage{xspace}

% For ensuring better copiability within pdfreader
\RequirePackage{cmap}

%% References and Links
\RequirePackage{hidelinks}[hyperref]
\daCheckAndLoadSetting{settings/pdfMetadata.tex}
```



D.1.2. Python Scripts

Listing D.2: Contents of fileTime.py.

```
import os

def getEditTime(filename):
    if os.path.isfile(filename):
        return os.stat(filename).st_mtime
    else:
        return 0
```

Listing D.3: Contents of createTempListingFile.py.

```
import os
import sys
import re
import time
import hashlib
import fileTime as ft

export_path = './TMP/lst/'

# Function to start the encoding
def run(sourcefile, lsttag, delim, force_rebuild):
    if not os.path.exists(export_path):
        os.makedirs(export_path)
    writefile = export_path + str(hashlib.sha256(sourcefile.encode()).hexdigest()) + '_' + lsttag + '.lst'
    if not force_rebuild and ft.getEditTime(sourcefile) < ft.getEditTime(writefile):
        return

    delim_reg = delim
    escapelist = ['+', '-', '!', '?', '[', ']', '{', '}']

    for character in escapelist:
        if character not in delim:
            pass
        else:
            delim_reg = delim_reg.replace(character, '\\' + character)

    if lsttag == 'all':
        tag = 'all'
    else:
        tag = delim + lsttag + delim

    expression_dictionary = {}
    expression_dictionary["anyTag"] = [re.compile('.*' + delim_reg + '[a-zA-Z0-9_]*' + delim_reg + '.*'), delim_reg + '[a-zA-Z0-9_]{1,64}' + delim_reg]
    expression_dictionary["newline"] = [re.compile('.*' + delim_reg + 'newline' + '[a-zA-Z0-9_]*' + lsttag + '\\b' + '[a-zA-Z0-9_]*' + delim_reg + '.*'),
```



docART Utility Handbook



```
delim_reg + 'newline ' + '[a-zA-Z0-9, ]*' + lsttag + '\\b' + '[a-zA-Z0-9, ]*' +  
delim_reg]  
expression_dictionary["comment"] = [re.compile('.*' + delim_reg + 'comment '  
+ '[a-zA-Z0-9, ]*' + lsttag + '\\b' + '[a-zA-Z0-9, ]*' + delim_reg + '.*'),  
delim_reg + 'comment ' + '[a-zA-Z0-9, ]*' + lsttag + '\\b' + '[a-zA-Z0-9, ]*' +  
delim_reg]  
  
outtext = fulltextSplit(sourcefile, lsttag, tag, expression_dictionary)  
lstofile=open(writefile,'w')  
lstofile.write(outtext)  
lstofile.close()  
  
def unindent(text):  
    if text == '':  
        return text  
  
    indents = True  
    while indents:  
        tmp = ''  
        lines = text.split('\n')  
        for l in lines:  
            if not l.startswith('\t') and l != '':  
                indents = False  
                break  
        if indents:  
            for l in lines:  
                tmp += l[1:]  
                if l != '':  
                    tmp = tmp + '\n'  
            text = tmp  
    return text  
  
def fulltextSplit(sourcefile, lsttag, tag, expression_dictionary):  
    sourcedata = open(sourcefile, 'r')  
    fulltext = sourcedata.read()  
    sourcedata.close()  
  
    if tag == 'all':  
        list = [fulltext]  
    else:  
        list = fulltext.split(tag)  
  
        if len(list) % 2 != 1 and len(list) > 1:  
            return 'Uneven number of Tags found!'  
  
        list = list[1::2]  
  
    outtext = ''  
    for paragraph in list:  
        paragraph = paragraph.rsplit('\n', 1)[0]#.rstrip()
```



```
linelist = paragraph.splitlines()
new_paragraph = ''
for line in linelist:
    if expression_dictionary["anyTag"][0].match(line) == None:
        new_paragraph += (line + '\n')
    elif expression_dictionary["newline"][0].match(line) != None:
        new_paragraph += '\n'
    elif expression_dictionary["comment"][0].match(line) != None:
        new_paragraph += (re.split(expression_dictionary["comment"][1],
line)[1] + '\n')

    outtext += (new_paragraph.rsplit('\n', 1)[0])

outtext = unindent(outtext)
return outtext.lstrip().rstrip()

if __name__ == '__main__':

    if len(sys.argv) < 4 or len(sys.argv) > 5:
        print('Wrong Arguments for Parsing Source Files!')
        print('Usage:')
        print('\t arg[1] = Path to Sourcefile')
        print('\t arg[2] = Tag Name')
        print('\t arg[3] = Tag Delimiter')
        print('\t arg[4] = Optional flag; "f" to rebuild all files')

    sourcefile = str(sys.argv[1])
    lsttag = str(sys.argv[2])
    delim = str(sys.argv[3])
    if len(sys.argv) == 5 and str(sys.argv[4]) == 'f':
        force_rebuild = True
    else:
        force_rebuild = False

    run(sourcefile, lsttag, delim, force_rebuild)
```

D.1.3. Lua Scripts

Listing D.4: Contents of typesetFigures.lua.

```
-- Definition of delimiters and keywords
local groupKey=";"
local elementKey=","

local missingImageKey="missingImage"
local widthKey="width"
local heightKey="height"
local aspectKey="aspect"
local gridKey="grid"
local layoutRowcolsMaxepKey = "x"
```




docART Utility Handbook



```
-- Definition of LaTeX command string elements
-- for figure environment
local figEnvStart = "\\begin{figure}[htb]\\centering"
local figEnvEnd = "\\end{figure}"

-- for subfigure environment
local subfigEnvStart = "\\begin{subfigure}{\"
local subfigEnvWidthEnd = \"}\\centering\"
local subfigEnvEnd = "\\end{subfigure}\"
local subfigHorizontalSpacer = "\\hfill\"
local subfigVerticalSpacer = "\\[0.25cm]\"
local subfigMaxUseableWidthFactor=0.99
local subfigDefaultHorizontalSpacerFactor=0.01
local subfigWidthAndHorizontalSpacerUnit = "\\linewidth\"

-- for graphics inclusion
local graphicsStart = "\\includegraphics[\"
local graphicsWidth = \"width=\"
local graphicsHeight = \"height=\"
local graphicsMiddle = \"]{\"
local graphicsEnd = \"}\"

-- split a string into a table
function splitStringIntoTable(stringToSplit, delimiter)
    result = {};
    for match in (stringToSplit .. delimiter):gmatch("(.-)\" .. delimiter) do
        table.insert(result, match);
    end
    return result;
end

-- return true when string contains other string
function stringRegex(stringToSearchIn, stringToFind)
    local matchResult = false

    for match in (stringToSearchIn):gmatch("(.-)\" .. stringToFind) do
        matchResult = true
    end

    return matchResult
end

-- determine the type

-- First stage of cleaning mandatory user input
function mandatoryUserInputCleaning(inputString)
    local inputRawArray= splitStringIntoTable(inputString, groupKey)-- split the
    string into table at ";\"
    local inputCleanedArray = {}

    for index=1, #inputRawArray-1 do -- len-1 to prevent empty content after
```



docART Utility Handbook



```
last ";"
    local stringToProcess = inputRawArray[index]
    local stringAsArray = {}
    local cleanedArrayStage1 = {}

    if stringToProcess ~= "" then -- do not process empty strings
        stringAsArray = splitStringIntoTable(stringToProcess, elementKey) --
        split the string into table at ","

        for subindex=1,#stringAsArray do
            if subindex == 1 then -- copy the image object (file path/
missingImage/TikZ) directly into clean array
                table.insert(cleanedArrayStage1,stringAsArray[subindex])
            -- else
            --     print("else")
            end
        end

        if #cleanedArrayStage1 ~= 0 then -- if the cleaning array is not
empty,
            table.insert(inputCleanedArray,cleanedArrayStage1) -- append it
to the precleaned array
        end
    end
end

return inputCleanedArray
end

function processMandatoryUserInput(inputString)
    local inputCleanedArray = mandatoryUserInputCleaning(inputString)
    local imageCodeArray = {}

    for index=1, #inputCleanedArray do
        if inputCleanedArray[index][1] ~= missingImageKey then -- code needs to
be added to support TikZ images
            local singleImageCodeGraphicsEnd = graphicsMiddle ..
inputCleanedArray[index][1] .. graphicsEnd
            singleImageCodeGraphicsEnd = singleImageCodeGraphicsEnd:gsub("{ ", "
{") -- fixing issues with white space after {
            local singleImageCodeArray = {graphicsStart,
singleImageCodeGraphicsEnd}
            table.insert(imageCodeArray,singleImageCodeArray)
        end
    end

    return {imageCodeArray}
end

function processOptionalUserInput(inputString)
    local layoutSpecifiedBool = false
```



docART Utility Handbook



```
local rowsMax = 1
local colsMax = 1
local totalNumberOfImages = 1

if inputString ~= "" then
    -- if more keywords/options allowed: first split at "," needs to be
    implemented
    stringAsArray = splitStringIntoTable(inputString, ",")

    if stringRegex(stringAsArray[1], gridKey) == true then
        layoutSpecifiedBool = true

        rowsMaxAndcolsMax = splitStringIntoTable(stringAsArray[2],
        layoutRowcolsMaxepKey)
        rowsMax = math.tointeger(rowsMaxAndcolsMax[1])
        colsMax = math.tointeger(rowsMaxAndcolsMax[2])
        totalNumberOfImages = rowsMax * colsMax
    end
end

return {layoutSpecifiedBool, rowsMax, colsMax, totalNumberOfImages}
end

function generateSubfigureCodeWithoutImages(layoutSpecifiedBool, rowsMax,
colsMax, numberOfImages, subfigDefaultWidthFactor)
    -- templates for the different subfigure end formatting options
    local singleSubfigNoSpacerTEMPLATEArray = {subfigEnvStart, subfigEnvWidthEnd
, subfigEnvEnd, ""}
    local singleSubfigVerticalSpacerTEMPLATEArray = {subfigEnvStart,
subfigEnvWidthEnd, subfigEnvEnd, subfigVerticalSpacer}
    local singleSubfigHorizontalSpacerTEMPLATEArray = {subfigEnvStart,
subfigEnvWidthEnd, subfigEnvEnd, subfigHorizontalSpacer}

    local modcolsMaxNumberOfImages = numberOfImages % colsMax

    local subfigCodeWithoutImagesArray = {}
    local indexCorrection = 0

    -- treat the special case first row, which might have different amount of
    cols
    -- due to even/odd number of images not fitting with the row/col layout
    if modcolsMaxNumberOfImages ~= 0 then
        -- number of images is not an integer multiple of columns
        -- -> less horizontal spacers/images and earlier line break than in the
        other rows
        for col=1, modcolsMaxNumberOfImages do
            if col < modcolsMaxNumberOfImages then
                if modcolsMaxNumberOfImages ~= 1 then
                    local customHspaceWidth = (1 - (modcolsMaxNumberOfImages *
subfigDefaultWidthFactor + subfigDefaultHorizontalSpacerFactor * (
modcolsMaxNumberOfImages-1)))/(colsMax) -- only works if no image width/
height specified
```



docART Utility Handbook



```
local customHspaceString = "\\hspace{" ..
subfigDefaultHorizontalSpacerFactor .. "\\linewidth}" -- customHspaceWidth

table.insert(subfigCodeWithoutImagesArray,{subfigEnvStart,
subfigEnvWidthEnd, subfigEnvEnd, customHspaceString})
else
    if rowsMax ~= 1 then
        table.insert(subfigCodeWithoutImagesArray,
singleSubfigVerticalSpacerTEMPLATEArray)
    else
        table.insert(subfigCodeWithoutImagesArray,
singleSubfigNoSpacerTEMPLATEArray)
    end
end
end
else
    if row ~= rowsMax then
        table.insert(subfigCodeWithoutImagesArray,
singleSubfigVerticalSpacerTEMPLATEArray)
    else
        table.insert(subfigCodeWithoutImagesArray,
singleSubfigNoSpacerTEMPLATEArray)
    end
end
end
end
else
    -- case when mod = 0 -> Number of images integer multiple of cols -> no
special formatting required
    for col=1, colsMax do
        if col < colsMax then
            table.insert(subfigCodeWithoutImagesArray,
singleSubfigHorizontalSpacerTEMPLATEArray)
        else
            if row ~= rowsMax then
                table.insert(subfigCodeWithoutImagesArray,
singleSubfigVerticalSpacerTEMPLATEArray)
            else
                table.insert(subfigCodeWithoutImagesArray,
singleSubfigNoSpacerTEMPLATEArray)
            end
        end
    end
end
end

-- general case for all other rows
for row=2, rowsMax do
    for col=1, colsMax do
        if col < colsMax then
            table.insert(subfigCodeWithoutImagesArray,
singleSubfigHorizontalSpacerTEMPLATEArray)
        else
            if row ~= rowsMax then
                table.insert(subfigCodeWithoutImagesArray,
```



docART Utility Handbook



```
singleSubfigVerticalSpacerTEMPLATEArray)
    else
        table.insert(subfigCodeWithoutImagesArray,
singleSubfigNoSpacerTEMPLATEArray)
    end
end
end
end
return subfigCodeWithoutImagesArray
end

-- function containing the logic to generate and print the LaTeX code
function typesetFigure(mandatoryUserInputRawString, optionalUserInputRawString,
    label, caption)
    local mandatoryData = processMandatoryUserInput(mandatoryUserInputRawString)
    local img = mandatoryData[1] -- get image array data

    local optionalData = processOptionalUserInput(optionalUserInputRawString)
    local layoutSpecifiedBool = optionalData[1] -- obtain information whether
user specified layout options

    local rowsMax = 1
    local colsMax = 1
    local numberOfImages = 1

    if layoutSpecifiedBool ~= false then -- if layout option specified
        -- copy rowsMax and colsMax from optional data
        rowsMax = optionalData[2]
        colsMax = optionalData[3]

        -- check which number is smaller to avoid issues with nil errors
        -- future: load missing graphics
        if optionalData[4] < #img then
            numberOfImages = optionalData[4]
        else
            numberOfImages = #img
        end
    else -- when no layout specified, set defaults (colsMax = 2, number of
images from mandatory input)
        colsMax = 2
        numberOfImages = #img

        local modcolsMaxNumberOfImages = numberOfImages % colsMax

        -- calculate the max number of rows considering odd/even number of
images
        rowsMax = (numberOfImages - modcolsMaxNumberOfImages)/colsMax +
modcolsMaxNumberOfImages
    end

    -- need to add code for width/height/aspect ratio calculation
```



```
-- hard coded as long as no specific styling of single elements allowed
local subfigDefaultWidthFactor = (subfigMaxUseableWidthFactor - (colsMax-1)
* subfigDefaultHorizontalSpacerFactor)/colsMax
local subfigDefaultWidthString = subfigDefaultWidthFactor ..
subfigWidthAndHorizontalSpacerUnit
local subfigWidth = {subfigDefaultWidthString}
local imgWidth = {"width=\\linewidth"}

local subfigureCodeArray = generateSubfigureCodeWithoutImages(
layoutSpecifiedBool, rowsMax, colsMax, numberOfImages,
subfigDefaultWidthFactor)

local commandString = figEnvStart -- initialize command string with start of
fig env

-- iterate over the complete image/subfigure data to assemble command string
for imgIndex=1, numberOfImages do
    -- create the strings for one subfigure with image content
    local subfigurePrefix = subfigureCodeArray[imgIndex][1] .. subfigWidth
[1] .. subfigureCodeArray[imgIndex][2]
    local subfigureSuffix = subfigureCodeArray[imgIndex][3] ..
subfigureCodeArray[imgIndex][4]
    local imageString = img[imgIndex][1] .. imgWidth[1] .. img[imgIndex][2]

    -- append them to the command string from the previous stage
    commandString = commandString .. subfigurePrefix .. imageString ..
subfigureSuffix
end

-- determine whether caption and/or label are provided
local captionAndLabelString = ""
if caption ~= "" then
    captionAndLabelString = captionAndLabelString .. "\\caption{" .. caption
.. "}"
    if label ~= "" then
        captionAndLabelString = captionAndLabelString .. "\\label{" .. label
.. "}"
    end
end

commandString = commandString .. captionAndLabelString .. figEnvEnd -- add
caption/label (if applicable) and terminate fig env
tex.sprint(commandString) -- print the assemble command to the TeX console
end
```

Listing D.5: Contents of typesetTables.lua.

```
-- function to determine number of columns, maximum rows in file and extract
header
function generateNumberOfColsMaxRowsAndRawHeader(filename)
    local maxRows = 0
```



docART Utility Handbook



```
local columns = 0
local rawHeader = ""
local tableHeader = ""

local input = io.open(filename, 'r') -- open the csv file locally
for line in input:lines() do -- go through the file line by line
    maxRows = maxRows + 1 -- increment the row counter to obtain the total
number of files after for loop has finished
    if maxRows == 1 then -- when first line is processed
        rawHeader = line -- set the current line as unprocessed header

        -- replace the "," from the csv file with nothing to get a string
which is similar to a lua function argument input
        -- then use select to determine the number of arguments passed to it
to obtain the final number of columns - 1
        columns= select(2, string.gsub(line, ",", "")) + 1
    end
end
input:close() -- close the file

-- replace the "," from the csv with " & " to be compliant with the LaTeX
table column seperator syntax;
-- "\\bfseries" ensures that the cell will be typeset in bold; note that
"\" required for escaping
rawHeader = rawHeader:gsub(",", " & \\bfseries ")

-- since replacing "," with " & \\bfseries " works not for the first column
(no "," available),
-- add "\\bfseries " manually to the beginning of the table header string
tableHeader = "\\bfseries " .. rawHeader

return columns, maxRows, tableHeader
end

function validateColAlign(colAlignSTRING, columns)
    local actualVal = 0
    local colAlignTABLE = {}
    -- local wrongSpecifiers = {} -- for improvement of error message in a later
version

    if colAlignSTRING ~= "" then -- when there was a column alignment specified
        if string.len(colAlignSTRING) == columns then -- if the length of the
column alignment string matches the number of columns
            for i = 1, string.len(colAlignSTRING) do
                table.insert(colAlignTABLE, string.sub(colAlignSTRING, i, i))
            end
        elseif string.len(colAlignSTRING) > columns then
            for i = 1, columns do
                table.insert(colAlignTABLE, string.sub(colAlignSTRING, i, i))
            end
        else
            for i = 1, columns do
```



docART Utility Handbook



```
        if i <= string.len(colAlignSTRING) then
            table.insert(colAlignTABLE,string.sub(colAlignSTRING, i, i))
        else
            table.insert(colAlignTABLE,"l")
        end
    end
end
else
    for i = 1, columns do
        table.insert(colAlignTABLE,"l")
    end
end

for col = 1, #colAlignTABLE do
    if colAlignTABLE[col] == "r" or colAlignTABLE[col] == "l" or
colAlignTABLE[col] == "c" then
        actualVal = actualVal + 1
    else
        -- table.insert(wrongSpecifiers,colAlignTABLE[col]) -- for
improvement of error message in a later version
        colAlignTABLE[col] = "l"
    end
end

local numberWrongSpecifier = columns - actualVal
local timesHelper = ""

if numberWrongSpecifier == 1 then
    timesHelper = numberWrongSpecifier .. " time"
else
    timesHelper = numberWrongSpecifier .. " times"
end

local warningMessage = "docART:WARNING: tables/columns/align: " ..
timesHelper .. " invalid column alignment option specified. Used \"l\"
instead. \n\n"

if numberWrongSpecifier ~= 0 then
    texio.write(warningMessage)
end

return colAlignTABLE
end

function generateCaptionAndLabel(label,caption, captionStatus)
    local captionAndLabelAbove = ""
    local captionAndLabelBelow = ""
    local captionAndLabel = ""
    local headCaption = ""
    local headCaptionPre = [[\tablename \]]
    local headCaptionPost = [[ continued from previous page.]]
end
```




docART Utility Handbook



```
if captionStatus ~= "nocaption" then
    headCaption = headCaptionPre .. [[ \thetable{}]] .. headCaptionPost
    captionAndLabel = [[\caption{}]] .. caption .. "}"
    if label ~= "" then
        captionAndLabel = captionAndLabel .. [[\label{}]] .. label .. "}"
    end
    if captionStatus == "captionabove" then
        captionAndLabelAbove = captionAndLabel .. "\\\\[0.5cm]"
    else
        captionAndLabelBelow = captionAndLabel
    end
else
    captionAndLabelAbove = ""
    captionAndLabelBelow = ""
    headCaption = headCaptionPre .. headCaptionPost
end

return captionAndLabelAbove, captionAndLabelBelow, headCaption
end

function setTableRowHighlightColors(captionStatus)
    if captionStatus ~= "nocaption" then
        tex.sprint([[\\rowcolors{2}{white}{tableRowHighlightColor}]] )
    else
        tex.sprint([[\\rowcolors{2}{tableRowHighlightColor}{white}]] )
    end
end

function parseAndGenerateTablePreambleAndMetric(filename, colAlignSTRING, label,
caption, captionStatus)
    columns, maxRows, tableHeader = generateNumberOfColsMaxRowsAndRawHeader(
filename)

    colAlignTABLE = validateColAlign(colAlignSTRING, columns)
    colAlignFINAL = table.concat(colAlignTABLE)

    captionAndLabelAbove, captionAndLabelBelow, headCaption =
generateCaptionAndLabel(label,caption, captionStatus)

    setTableRowHighlightColors(captionStatus)

    tex.sprint("\\def\\filename{" .. filename .. "}")
    tex.sprint("\\def\\columns{" .. columns .. "}")
    tex.sprint("\\def\\maxRows{" .. maxRows .. "}")
    tex.sprint("\\def\\colAlign{" .. colAlignFINAL .. "}")
    tex.sprint("\\def\\tableHeader{" .. tableHeader .. "}")
    tex.sprint("\\def\\captionAndLabelAbove{" .. captionAndLabelAbove .. "}")
    tex.sprint("\\def\\captionAndLabelBelow{" .. captionAndLabelBelow .. "}")
    tex.sprint("\\def\\headCaption{" .. headCaption .. "}")
    tex.sprint("\\tabulinesep=1.2mm") -- for better spacing around lines
    tex.sprint("\\begin{longtable}[c]" .. colAlignFINAL .. "}")
end
```



```
function parseAndGenerateTableData(filename,maxRows)
    local counter = 0
    local tableData = ""

    local input = io.open(filename, 'r')
    for line in input:lines() do
        if counter > 0 then
            tableData = line
            tableData=tableData:gsub(",", " & ")
            if counter < maxRows-1 then
                tex.sprint(tableData .. " \\\\")
            else
                tex.sprint(tableData)
            end
        end
        counter = counter + 1
    end
    input:close()
end
```

D.2. docART Utility Handbook

D.2.1. docART Theme Files

Listing D.6: Content of the default docART theme file for color settings.

```
% docART color theme
\colorlet{docartOrange}{orange}
\definecolor{docartTurquoise}{HTML}{008983}

% Colors for text
\colorlet{headlinecolor}{docartTurquoise}%{HTML}{008983}

% Colors for tables
\definecolor{tableRowHighlightColor}{gray}{0.9}

% Colors for highlight boxes
\definecolor{warningColor}{HTML}{ffd42a}
\definecolor{infoColor}{HTML}{0b96fb}
```

Listing D.7: Content of the default docART theme file for figures.

```
\NewDocumentCommand{\daDefaultFigureTEMPLATE}{0{} +mmm}{%
    \directlua{%
        local typesetFig=require("./docart-utility/scripts/luatypesetFigures")
        local optionalUserInputRawString = "\luaescapestring\detokenize{#1}"
        local mandatoryUserInputRawString = "\luaescapestring\detokenize{#2}"
        local label = "\luaescapestring\detokenize{#3}"
```



docART Utility Handbook



```
        local caption = "\luaescapestring{\detokenize{#4}}"
        %
        typesetFigure(mandatoryUserInputRawString, optionalUserInputRawString,
        label, caption)
    }
}

\NewDocumentCommand{\daFigureDefaultCaptionOff}{0{} +m}{%
    \daDefaultFigureTEMPLATE[#1]{#2}{}{}}
}

\NewDocumentCommand{\daFigureDefaultCaptionBelow}{0{} +mmm}{%
    \daDefaultFigureTEMPLATE[#1]{#2}{#3}{#4}}
}
```

Listing D.8: Content of the default docART theme file for font settings.

```
\setmainfont[%
    Path=./docart-utility/fonts/,%
    BoldFont={texgyreheros-bold.otf},%
    ItalicFont={texgyreheros-italic.otf},%
]{texgyreheros-regular.otf}

\setmonofont[%
    Path=./docart-utility/fonts/,%
    BoldFont={Hack-Bold.ttf},%
    ItalicFont={Hack-Italic.ttf},%
]{Hack-Regular.ttf}
```

Listing D.9: Content of the default docART theme file for footer settings.

```
\ifoot{%
    \normalfont %
    \scalebox{0.9}{%
        \small \color{gray}\productName~Utility, \productVersion, \docRevText %
    }%
}

\ofoot{%
    \normalfont %
    \scalebox{0.9}{%
        \small \color{gray} License: %
        \href{https://www.gnu.org/licenses/gpl-3.0.en.html}{GPLv3}%
    }%
}

\cfoot{%
    \normalfont\thepage %
}
```

Listing D.10: Content of the default docART theme file for header settings.

```
\ihead{}
\ohead{}
\chead{
```



docART Utility Handbook



```
\tikz{
  \node[inner sep=0, outer sep=0, anchor=west](logo) at (0,0){
    \includegraphics[width=1.5cm]{docart-utility/class-master-theme/
cmt-images/docart-logo_folder_v0.pdf}
  };
  \node[text=orange, anchor=west, align=center, font=\normalfont](text) at
($ (logo.east)+(5em,0)$){\Large\bfseries \productName~\Large\bfseries Utility
Handbook};
  \node[inner sep=0, outer sep=0, anchor=west](logo2) at ($ (text.east)+(5em
,0)$){
    \includegraphics[width=1.5cm]{docart-utility/class-master-theme/
cmt-images/docart-logo_folder_v0.pdf}
  };
}
```

Listing D.11: Content of the default docART theme file for highlight box settings.

```
\newenvironment{daWarningBox}{
  \rowcolors{2}{white}{white} % necessary for new longtables to prevent gray
boxes
  \par\addvspace{\baselineskip}
  \tabularx{\linewidth}{c!{\color{warningColor} \vrule width 2 pt}X}% l
  \begin{minipage}[t]{0.05\linewidth}
    \centering
    \raisebox{-1em}{
      \includegraphics[width=2em]{docart-utility/class-master-theme/cmt-images
/warning.png}
    }
  \end{minipage}
  &
  \begin{minipage}[t]{0.8\linewidth}
}{
  \end{minipage}
  \endtabularx
  \par\addvspace{\baselineskip}
}

\newenvironment{daInfoBox}{
  \rowcolors{2}{white}{white} % necessary for new longtables to prevent gray
boxes
  \par\addvspace{\baselineskip}
  \tabularx{\linewidth}{c!{\color{infoColor} \vrule width 2 pt}X}% l
  \begin{minipage}[t]{0.05\linewidth}
    \centering
    \raisebox{-1em}{
      \includegraphics[width=1.75em]{docart-utility/class-master-theme/cmt-images/
info_blue.png}
    }
  \end{minipage}
  &
  \begin{minipage}[t]{0.8\linewidth}
}{
  \end{minipage}
  \endtabularx
  \par\addvspace{\baselineskip}
}
```



docART Utility Handbook



```
\end{minipage}  
\endtabularx  
\par\addvspace{\baselineskip}  
}
```

Listing D.12: Content of the default docART theme file for icon settings.

```
\newcommand{\daIconTEMPLATE}[2][0.75em]{\includegraphics[height=#1]{#2}}  
\newcommand{\daTestIcon}{\daIconTEMPLATE{./docart-utility/class-master-theme/cmt-  
-images/testicon.png}\xspace}  
  
\newcommand{\daIcon}[2][12pt]{  
  \setlength{\logowidth}{#1}  
  \ifstrequal{#2}{warning}%  
  {\daIconTEMPLATE{./docart-utility/class-master-theme/cmt-images/warning}}% #  
  {}%  
  \ifstrequal{#2}{info}%  
  {\daIconTEMPLATE{./docart-utility/class-master-theme/cmt-images/info_blue}}%  
  {}%  
  \ifstrequal{#2}{varinfo}%  
  {\scalebox{0.9}{\bcinfo}}%  
  {}%  
  \ifstrequal{#2}{varwrench}%  
  {\bcoutil}%  
  {}%  
  \ifstrequal{#2}{varpaperclip}%  
  {\bctrombone}%  
  {}%  
  \ifstrequal{#2}{docART}%  
  {\daIconTEMPLATE{./docart-utility/class-master-theme/cmt-images/docart_icon  
}}%  
  {}  
  \xspace  
}
```

Listing D.13: Content of the default docART theme file for listing settings.

```
\newcommand{\listingCaptionsContentSetting}{custom}  
  
\NewDocumentCommand{\daListingTEMPLATE}{0{} +mmm}{%  
  \directlua{%  
    %%%%%%%%% Loading external data to lua local  
    %%%%%%%%%  
    local sha = require("./docart-utility/scripts/lua/sha2")% loading  
    hashing library  
    local settings = "\luaescapestring{\detokenize{#1}}" % detokenize/escape  
    settings argument  
    local filename = "#2" % copy code filename to lua  
    local label = "\luaescapestring{\detokenize{#4}}" % detokenize/escape  
    label argument  
    local caption = "\luaescapestring{\detokenize{#5}}" % detokenize/escape  
    caption argument  
    local regexLabel = "#3" % copy code regex label to lua
```



docART Utility Handbook



```
local regexLabelDelim = "\luaescapestring{\daTagDelim}" % copy label
delim to lua
%%%%%%%%%%%% Hardcode output directory and output file ending
%%%%%%%%%%%%
local lstPath = "./TMP/lst/" % setting output directory
local fileEnding = ".lst" % setting output file ending
%%%%%%%%%%%% Generate hash, ofilename and command strings
%%%%%%%%%%%%
local hashResult = sha.sha256(filename) % hashing the code file name
local ofilename = lstPath .. hashResult .. "_" .. regexLabel ..
fileEnding
local pythonCommand = "\\ShellEscape{python ./docart-utility/scripts/
python/createTempListingFile.py " .. filename .. " " .. regexLabel .. " " ..
regexLabelDelim .. "}" % constructing the python call
local lstCommand = "\\lstinputlisting[caption={ " .. caption .. " }, label
={ " .. label .. " }" .. settings .. " ]{ " .. ofilename .. " }" % constructing the
LaTeX listing command
%%%%%%%%%%%% Print and execute command strings
%%%%%%%%%%%%
tex.sprint(pythonCommand) % print and let LaTeX execute python command
tex.sprint(lstCommand) % print and let LaTeX execute listing command
}%
}%

\NewDocumentCommand{\daListingCaptionAbove}{0{ } +mmm}{%
  \daListingTEMPLATE[,captionpos=t, #1]{#2}{#3}{#4}{#5}
}

\NewDocumentCommand{\daListingCaptionBelow}{0{ } +mmm}{%
  \daListingTEMPLATE[,captionpos=b, #1]{#2}{#3}{#4}{#5}
}

\NewDocumentCommand{\daListingCaptionOff}{0{ } +mm}{%
  \directlua{%
    %%%%%%%%% Loading external data to lua local
    %%%%%%%%%
    local sha = require("./docart-utility/scripts/lua/sha2") % loading
    hashing library
    local settings = "\luaescapestring{\detokenize{#1}}" % detokenize/escape
    settings argument
    local filename = "#2" % copy code filename to lua
    local regexLabel = "#3" % copy code regex label to lua
    local regexLabelDelim = "\luaescapestring{\daTagDelim}" % copy label
    delim to lua
    %%%%%%%%% Hardcode output directory and output file ending
    %%%%%%%%%
    local lstPath = "./TMP/lst/" % setting output directory
    local fileEnding = ".lst" % setting output file ending
    %%%%%%%%% Generate hash, ofilename and command strings
    %%%%%%%%%
    local hashResult = sha.sha256(filename) % hashing the code file name
```



docART Utility Handbook



```

    local ofilename = lstPath .. hashResult .. "_" .. regexLabel ..
fileEnding
    local pythonCommand = "\\ShellEscape{python ./docart-utility/scripts/
python/createTempListingFile.py " .. filename .. " " .. regexLabel .. " " ..
regexLabelDelim .. "}" % constructing the python call
    local lstCommand = "\\lstinputlisting[" .. settings .. "]{\".. ofilename ..
"}" % constructing the LaTeX listing command
    %%%%%%%%% Print and execute command strings
%%%%%%%%%%%%
    tex.sprint(pythonCommand) % print and let LaTeX execute python command
    tex.sprint(lstCommand) % print and let LaTeX execute listing command
}%
}%

\\newcommand{\\daListingInline}[2][{}{
  \\lstinline[%
    %columns=fixed,%
    %backgroundcolor=\\color{daListingColorBackground},
    commentstyle=\\color{daListingColorComment},
    directivestyle={\\color{lime}\\bfseries},
    numberstyle=\\tiny\\color{daListingColorLineNumbers},
    stringstyle={\\bfseries\\color{daListingColorString}},
    basicstyle=\\ttfamily\\footnotesize,
    #1%
  ]$#2$ % additional settings get ignored???
  \\xspace
} % does not accept LaTeX macros due to \\; \\lstinline$$ has to be used manually

```

Listing D.14: Content of the default docART theme file for table settings.

```

\\def\\tableborder{1pt}

\\NewDocumentCommand{\\defaultFancyTableTEMPLATE}{mmmmm}{
  \\directlua{%
    local colAlignSTRING = "#1"
    local filename = "\\luaescapestring{\\detokenize{#2}}"
    local label = "\\luaescapestring{\\detokenize{#3}}"
    local caption = "\\luaescapestring{\\detokenize{#4}}"
    local captionStatus = "#5"
    %
    local typeset = require("./docart-utility/scripts/luatypesetTables")
    parseAndGenerateTablePreambleAndMetric(filename, colAlignSTRING, label,
caption, captionStatus)
  }
  % begin of longtable environment (needs to be printed, otherwise
colAlign cannot be set procedurally)
  \\captionAndLabelAbove
  \\rowcolor{white}
  \\tableHeader\\
  \\midrule
  \\endfirsthead
  \\rowcolor{white}

```



docART Utility Handbook



```

\multicolumn{\columns}{c}{\small\color{gray} \headCaption}\[0.25cm]
\rowcolor{white}
\tableHeader\\
\midrule
\endhead
\midrule
\rowcolor{white}
\multicolumn{\columns}{c}{\small\color{gray}Table continued on next page
.}\}
\endfoot
\rowcolor{white}
\midrule
\rowcolor{white}
\captionAndLabelBelow
\endlastfoot
\directlua{%
    local filename = "\filename"
    local maxRows = "\maxRows"
    local typeset = require("../docart-utility/scripts/lua/typesetTables
")
    parseAndGenerateTableData(filename,maxRows)
}%
\end{longtable}
} % #1 = col algin data, #2 = filename, #3 = label, #4 = caption, #5 = caption
status, #6 = column modifiers (not implemented yet)

\NewDocumentCommand{\daTableDefaultFancyCaptionOff}{0{default} m}{
    \defaultFancyTableTEMPLATE{#1}{#2}{}{}{nocaption}{}
}

\NewDocumentCommand{\daTableDefaultFancyCaptionBelow}{0{default} mmm}{
    \defaultFancyTableTEMPLATE{#1}{#2}{#3}{#4}{captionbelow}{}
}

\NewDocumentCommand{\daTableDefaultFancyCaptionAbove}{0{default} mmm}{
    \defaultFancyTableTEMPLATE{#1}{#2}{#3}{#4}{captionabove}{}
}

\newcommand{\daTableCodeDescriptionCaptionBelow}[3]{
    \begin{table}[h!]
    \centering
    \pgfplotstabletypeset[%
        col sep=comma,%
        header=false,
        every head row/.style={
            output empty row,
            before row={%
                \textbf{data type} & \textbf{description}\\
            },
            after row={%
                \noalign{\hrule height \tableborder}%
            },

```




```
},
columns/0/.style={%
string type,%
column type=r,%
postproc cell content/.append style={
/pgfplots/table/@cell content/.add={\lstinline$}{$$},
},
},
columns/1/.style={string type, column type=l},
%every even row/.style={before row={\rowcolor[gray]{0.9}}},%
]{#1}
\caption{#3}
\label{#2}
\end{table}
}
```

Listing D.15: Content of the default docART theme file for title page settings.

```
\newcommand{\daGenerateTitlepage}{
\begin{titlepage}
\centering
\daTitlepageTitle
\daTitlepageLogo
\vfill
\daTitlepageDescription
\vfill
\daTitlepageAuthor
\vfill
\daTitlepageDocumentInformation
\daTitlepageLicense
\end{titlepage}
}
```

D.2.2. User Settings

Listing D.16: Content of the docART Utility Handbook settings file: colorSettings.tex.

```
%% Colors
\definecolor{daListingColorBackground}{rgb}{0.72,0.72,0.69}
\definecolor{daListingColorLineNumbers}{rgb}{0.5,0.5,0.5}
\definecolor{daListingColorComment}{rgb}{0.33,0.33,0.33}
\definecolor{daListingColorString}{rgb}{0.87,0.07,0.27}
```

Listing D.17: Content of the docART Utility Handbook settings file: listingsSettings.tex.

```
\def\daTagDelim{+++} % required for regex parsing of in block code listing

\lstdefinestyle{C++}{
language=[11]c++,
basicstyle={\ttfamily\footnotesize},
backgroundcolor={\color{daListingColorBackground}},
commentstyle={\color{daListingColorComment}},
```



```
directivestyle={\color{orange}\bfseries},
numberstyle={\tiny\color{daListingColorLineNumbers}},
stringstyle={\color{daListingColorString}},
breakatwhitespace=false,
breaklines=true,
keepspaces=true,
numbers=none,
numbersep=5pt,
showspaces=false,
showstringspaces=false,
showtabs=false,
tabsize=4,
keywordstyle={\color{blue}\ttfamily\bfseries},
keywordstyle=[2]{\color{docartTurquoise}\ttfamily\bfseries},
keywordstyle=[3]{\color{orange}\ttfamily\bfseries},
keywordstyle=[4]{\color{red}\ttfamily\bfseries},
morekeywords={int, float, double},
morekeywords=[2]{std},
morekeywords=[3]{MB_OK, once},
morekeywords=[4]{NULL, Run, Shutdown}
}

\lstdefinestyle{Python}{
  language=Python,
  backgroundcolor=\color{daListingColorBackground},
  commentstyle=\color{daListingColorComment},
  directivestyle={\color{orange}\bfseries},
  numberstyle=\tiny\color{daListingColorLineNumbers},
  stringstyle=\color{daListingColorString},
  basicstyle=\ttfamily\footnotesize,
  breakatwhitespace=false,
  breaklines=true,
  keepspaces=true,
  numbers=none,
  numbersep=5pt,
  showspaces=false,
  showstringspaces=false,
  showtabs=false,
  tabsize=4,
  procnamekeys={def}, % bonus, for function names
  procnamestyle=\bfseries\color{cyan},
}

\lstdefinestyle{LaTeX}{
  language=[LaTeX]TeX,
  backgroundcolor=\color{daListingColorBackground},
  commentstyle=\color{daListingColorComment},
  directivestyle={\color{lime}\bfseries},
  numberstyle=\tiny\color{daListingColorLineNumbers},
  stringstyle={\bfseries\color{daListingColorString}},
  basicstyle=\ttfamily\footnotesize,
  keywordstyle = {\bfseries\color{blue}},
}
```



docART Utility Handbook



```
keywordstyle = [2]{\bfseries\color{orange}},
breakatwhitespace=false,
breaklines=true,
keepspaces=true,
numbers=none,
numbersep=5pt,
showspaces=false,
showstringspaces=false,
showtabs=false,
tabsize=4,
morekeywords={
  \tableofcontents,\lstset,\chapter, \section, \subsection,\listoftodos, \
foreach, \missingfigure,\enquote, \textcolor, \lstinline,\ProvidesClass,\
LoadClass,\RequirePackage,
  \IfFileExists, \ClassError,\usetikzlibrary,\setkomafont,
  \captionsetup,\pgfplotsset,\RedeclareSectionCommand,\definecolor,\
includegraphics,\setmainfont,\setmonofont,\ifoot,\ofoot,\cfoot,\scalebox,\
color,\href, \ihead, \chead, \ohead, \automark, \pagemark, \colorlet, \
currenttime, \NewDocumentCommand, \directlua, \luaescapestring, \detokenize,
\tikz, \node, \draw, \fill, \filldraw, \shade, \shadedraw, \path, \tabularx,
\endtabularx, \rowcolors, \setlength, \ifstrequal, \bcinfo, \bcoutil, \
bctrombone, \rowcolor, \xspace, \midrule, \endfirsthead, \endhead, \endfoot,
\endlastfoot, \pgfplotstablotype, \lstdefinestyle, \hypersetup
},
morekeywords = [2]{
  daWarningBox,
  daInfoBox,
  \tableHeader,
  \captionAndLabelBelow,
  \daTableDefaultFancyCaptionOff,
  \daTableDefaultFancyCaptionBelow,
  \daTableDefaultFancyCaptionAbove,
  \daListingInline,
  \daDefaultFigureTEMPLATE,
  \daIconTEMPLATE,
  \daIcon,
  \daListingCaptionBelow,
  \daListingCaptionAbove,
  \daListingCaptionOff,
  \daTableCodeDescriptionCaptionBelow,
  \productVersion,
  \productName,
  \captionContent,
  \captionAndLabelAbove,
  \fileName,
  \daTestIcon,
  \daCheckAndLoadSetting,
  \docRevText,
  \releaseState,
  \releaseDate,
  \documentRevision,
  \daGenerateTitlepage,
```



```
\daListingInline,
\daListingTEMPLATE,
\defaultFancyTableTEMPLATE,
\daTagDelim,
\daTitlepageTitle,
\daTitlepageLogo,
\daTitlepageDescription,
\daTitlepageAuthor,
\daTitlepageDocumentInformation,
\daTitlepageLicense,
\daFigureDefaultCaptionOff,
\daFigureDefaultCaptionBelow
},
% keywordstyle=\color{blue}\ttfamily\bfseries,
}

\lstdefinestyle{Lua}{
  language=[5.3]Lua,
  backgroundcolor=\color{daListingColorBackground},
  commentstyle=\color{daListingColorComment},
  directivestyle={\color{lime}\bfseries},
  numberstyle=\tiny\color{daListingColorLineNumbers},
  stringstyle={\bfseries\color{daListingColorString}},
  basicstyle=\ttfamily\footnotesize,
  keywordstyle = {\bfseries\color{blue}},
  keywordstyle = [2]{\bfseries\color{orange}},
  breakatwhitespace=false,
  breaklines=true,
  keepspaces=true,
  numbers=none,
  numbersep=5pt,
  showspace=false,
  showstringspaces=false,
  showtabs=false,
  tabsize=4,
  morekeywords={for, end, local}
}

\lstset{style=C++}
```

Listing D.18: Content of the docART Utility Handbook settings file: pdfMetadata.tex.

```
\hypersetup{
  pdftitle={\productName~Utility Handbook},
  pdfsubject={Handbook for the \productName~Utility, \productVersion, \
docRevText},
  pdfauthor={Martin Stimpfl, Daniel Sacré},
  pdfkeywords={Handbook, Documentation, Software Development, Hardware
Development, \LaTeX, \productName~Utility},
  pdfcreator={docART Utility, \productVersion; Backend: (Lua)\LaTeX~with
hyperref}
}
```



docART Utility Handbook



Listing D.19: Content of the docART Utility Handbook settings file: `projectDetails.tex`.

```
\def\productName{docART}
\newcommand{\releaseState}{alpha}
\newcommand{\releaseDate}{2022-04-30}
\newcommand{\productVersion}{\releaseState-\releaseDate}
\newcommand{\documentRevision}{0}
\newcommand{\docRevText}{Doc. Rev. \documentRevision}
```

Listing D.20: Content of the docART Utility Handbook settings file: `titlepageContent.tex`.

```
\def\daTitlepageTitle{
  \begin{tikzpicture}
    \node[yshift=-5cm,xslant=1,yscale=1.6,orange!40,opacity=.2] at
    (10.5,-6.5) {\scalebox{6}{\productName}};
    \node[yshift=-5cm,orange] at (9,-7) {\scalebox{6}{\productName}};
  \end{tikzpicture}
  \phantom{}\\[3cm]
}

\def\daTitlepageLogo{
  \begin{tikzpicture}
    \node[inner sep=0, outer sep=0,anchor=west](logo) at (0,0){
      \includegraphics[width=5cm]{docart-utility/class-master-theme/
      cmt-images/docart-logo_folder_v0.pdf}
    };
  \end{tikzpicture}
}

\def\daTitlepageDescription{
  \textcolor{orange}{
    \large A Python/Lua(\LaTeX) Utility and \LaTeX~Class\\[0.125cm]
    for Semi-automated Documentation\\[0.125cm]
    of Projects involving Hardware and/or Software.
  }
}

\def\daTitlepageAuthor{
  \textcolor{gray}{\large Authors: Martin Stimpfl, Daniel Sacré}\\[0.125cm]
}

\def\daTitlepageDocumentInformation{
  \textcolor{darkgray}{\Huge -- Handbook --}\\[1cm]
  \textcolor{gray}{\productName~Utility, \productVersion}\\
  \textcolor{gray}{Document Revision \documentRevision}\\[0.5cm]
}

\def\daTitlepageLicense{
  \footnotesize
  \textcolor{gray}{License: GNU General Public License (GPLv3)}
}
```



D.2.3. Content Examples

Listing D.21: Listing of the complete main \LaTeX file required to generate the documentation

```
% !TeX program = lualatex

\documentclass{docart-utility/docart}

\begin{document}
  \daGenerateTitlepage

  \tableofcontents

  \input{./document_parts/docart_documentation_preface_v0.tex}
  \input{./document_parts/docart_documentation_aim-and-scope_v0.tex}

  \input{./document_parts/docart_documentation_legal-and-licenses_v0.tex}

  \input{./document_parts/docart_documentation_installation_v0.tex}
  \input{./document_parts/docart_documentation_config_v0.tex}

  \newpage
  \input{./document_parts/docart_documentation_usage_v0.tex}

  \newpage
  \input{./document_parts/docart_documentation_basic-functionality_v0.tex}

  \newpage
  \input{./document_parts/docart_documentation_adapt-default-theme_v0.tex}

  \newpage
  \input{./document_parts/docart_documentation_known-bugs_v0.tex}

  \input{./document_parts/docart-documentation_appendix_v0.tex}

\end{document}
```

Listing D.22: Listing of the \LaTeX code to generate the basic functionality chapter of the documentation

```
\chapter{\productName: Basic Functionality}
  \input{./document_parts/basic_functionality_parts/docart_documentation_bf_
highlight-boxes_v0.tex}

  \newpage
  \input{./document_parts/basic_functionality_parts/docart_documentation_bf_
icons_v0.tex}

  \newpage
  \input{./document_parts/basic_functionality_parts/docart_documentation_bf_
figures_v0.tex}

  \newpage
```



docART Utility Handbook



```
\input{./document_parts/basic_functionality_parts/docart_documentation_bf_
tables_v0.tex}

\newpage
\input{./document_parts/basic_functionality_parts/docart_documentation_bf_
listings_v0.tex}

\newpage
\input{./document_parts/basic_functionality_parts/docart_documentation_bf_
labels_v0.tex}
```

Listing D.23: Listing of the \LaTeX code to generate the “Highlight Boxes” section

```
\section{Highlight Boxes}
Sometimes it is necessary to highlight the importance of information.
Changing the font type to bold or italics is not the best way. On the
one hand, it breaks the reading flow and on the other hand font changes
can be too subtle if the reader is just scanning the document very
rapidly.
% Generating a warning box.
\begin{daWarningBox}
    This is a box designed to indicate a warning. Its design is by
    purpose very minimalistic, but can be altered to suit any needs.
    Additionally, it does not use the complete line width to provide a
    clear visual separation between continuous text and the boxes.
\end{daWarningBox}
These boxes should only contain very little text and used sparsely;
otherwise the highlighting effect wears off. By design, these boxes do
not provide pagebreak functionality, which is intentional to force the
user to fill these boxes with only the necessary content.
\newline The highlight box above was created using the following  $\text{\LaTeX}$ ~
code:
\lstset{style=LaTeX}
\daListingCaptionBelow{./document_parts/basic_functionality_parts/docart_
documentation_bf_highlight-boxes_v0.tex}{warningBoxExample}{lst:bf:highlight-
boxes:warningBox}{%
    The  $\text{\LaTeX}$ ~code required for generating a warning box. %
}

In  $\text{\productVersion}$ ,  $\text{\productName}$ ~provides a second custom environment
called  $\text{\daListingInline{daInfoBox}}$ . Replacing  $\text{\daListingInline{daWarningBox}}$ 
with  $\text{\daListingInline{daInfoBox}}$  in the code snippet
above, leads to the following output:
\begin{daInfoBox}
    This is a box designed to indicate important information. Its design
    is by purpose very minimalistic, but can be altered to suit any
    needs. Additionally, it does not use the complete line width to
    provide a clear visual separation between  $\text{\mbox{continuous}}$  text
    and the boxes.
\end{daInfoBox}
```

Listing D.24: Listing of the \LaTeX code to generate the “Icons in Text” section



docART Utility Handbook



`\section{Icons (\daTestIcon) in Text}`

In comparison to many other word processors, the `\productName~Backend` allows inclusion of custom icons into the continuous text. It also works for headings, tables, captions and lists. The icons can be either images (png, jpg, pdf), TikZ-Elements and more. There is also a native scalability with font size, as it is demonstrated below with some of the default icons the `\productName~Utility` ships with. `%\[0.5cm]`

`\begin{daInfoBox}`

For version `\productVersion`, the authors do not recommend the user to create custom icons and therefore do not provide a how-to-guide.

The reason being that in one of the next releases, substantial changes will be made to the icons Backend, which should make the icon creation by the end user way easier and more consistent.

`\end{daInfoBox}`

This is a test normal sized text with an `\daTestIcon` icon within the text `.\[0.125cm]`

This is is a normal sized text with an icon at the end `\daTestIcon.\[0.125cm]`

`{\large` This is a large text with an `\daTestIcon` icon within the text `.\[0.125cm]`

`{\Large` This is a Large text with an `\daTestIcon` icon within the text `.\[0.125cm]`

`{\LARGE` This is a LARGE text with an `\daTestIcon` icon within the text `.\[0.125cm]`

`\lstset{style=LaTeX}`

`\daListingTEMPLATE{./document_parts/basic_functionality_parts/docart_documentation_bf_icons_v0.tex}{bficons}{lst:bf:icons-in-text:icon-example}{%\LaTeX~code for using icons in the text.}`

The `\lstineline\daTestIcon` comes in handy if a new font/typeface should be assessed.

`\newline` For everyday use,

`\productName~`provides a selection of default icons. The icons can be accessed by calling the macro `\lstineline[emphstyle={\color{orange}\bfseries},moreemph={icon}]$\daIcon{ICONNAME}$`. For example,

`\lstineline[emphstyle={\color{orange}\bfseries},moreemph={icon}]$\daIcon{warning}$`

generates an exclamation mark within a yellow triangle `\daIcon{warning}`.

Icons with names containing `\enquote{\lstinelinevar}` can be scaled manually.

For example, `\lstineline[emphstyle={\color{orange}\bfseries},moreemph={icon}]$\daIcon[10pt]{varwrench}$` generates a wrench `\daIcon[10pt]{varwrench}` of width `\mbox{10\,pt}`. The table `\ref{tab:bf:iconOverview}` lists all the currently available icons in `\mbox{\productName~\mbox{\productVersion}}`.

`\daTableDefaultFancyCaptionAbove{./tables/docART_icon-overview.csv}{tab:bf:iconOverview}{Overview of available icons in \productName~\productVersion.}`