The SKB package - Create and maintain a repository for long-living documents

Sven van der Meer

2010-08-04 v0.5

Abstract

This package provides macros that help to build a repository for long living documents. It focuses on structure and re-use of text, code, figures etc. The basic concept is to first separate structure from content (i.e. text about a topic from the structure it is presented by) and then separating the content from the actual published document, thus enabling easy re-use of text blocks in different publications (i.e. text about a protocol in a short article about this protocol as well as in a book about many protocols); all without constantly copying or changing text. As a side effect, using the document classes provided, it hides a lot of LaTeX from someone who just wants to write articles and books.

Contents

1 The Intent

Provide a LaTeXpackage that helps to create and maintain a repository for long-living documents. It's probably not usefull for some short-term articles, however, I learned that most of my short term articles eventually become part of my long-term documents. Here you go. The repository should allow for easy access to 'stuff': text blocks, senteces, paragraphs, sections, complete chapters. But also to figures, code sniplets, examples, etc. And I do want to limit the amount of repetition of information. For example, if I use a certain example in an article I might want to use the same (identical) example in a book or a presentation or lecture notes. If I only copy the example I have to maintain several sources, and over time I will not remember which of them is normative. As a side effect, I also want to optimise document creation and limit the LaTeXor document class specific code in my documents.

2 The Story

2.1 The Short Story

I have written papers, done a lot of presentations, provided some book chapters, still working on a book, participated in many research proposals and projects, and created tons of notes and figures. As of early 2009, most of that information was distributed over the repositories of different projects and organisations I worked for, in some document management systems, on several websites, databases, my preferred email client (which changed twice), different computers and later even different external hard drives and USB sticks. Looking for specific text or a particular figure could easily end in a days work. Tools like desktop search engines can help to find 'stuff'. I used them, but if they found anything it was hard to maintain the context it was written in and some formats or sources were out of reach for them. Even worse with figures and the many versions some of them evolved in over time. After multiple jobs and several years, all I had is kind of a very messy base of knowledge, well-hidden somewhere, thus very difficult to locate and impossible to maintain.

So I started early 2009 to re-organise my 'stuff'. At the same time, I did realise that moving away from LaTeXwas part of the problem (and I thought using the other text processor would help, it actually didn't, long-term). So LaTeXbecame,

again, the text processor of choice, and with it the ability for a complete different approach to organise my 'stuff'. This was the moment the SKB was created. SKB stands for Sven's Knowledge Base. The LaTeXpackage `skb`, described in this article, forms part of a larger software system that uses SQLite databases, a small PHP framework, Apache for HTML access and recently also a Java port.

My document repository uses the `skb` package, so most of my documents are eventually LaTeXdocuments. I am saying eventually because I still use other tools (like Microsoft's Powerpoint), but integrate their output in my repository. I do all my figures these days using Inkscape, so the source is SVG and the output for LaTeXdocuments PDF. For editing the text files I do flip between UE Studio and LeD. Parts of the content (such as acronyms and bibliographic information) are maintained in SQLite databases and exported to LaTeX. This package now shows how I build my documents.

2.2 The Long Story

Over several years of writing documents (articles, books, reports, standards, research proposals) ideas and concepts became distributed (actually a euphemism for 'hidden') within many many documents (in all sorts of formats) located at many many locations (such as local file system, document management system, subversion/perforce systems, web servers, email clients). The problems associated to this situation are manifold:

- Ideas/concepts are hidden, often un-accessible and, as I experienced, search tools are of limited help.
- The documents are written in all sorts of formats or available only in (usually proprietary) binary formats. Ever tried to open a document written in MS WinWord 6.0 with customised document template in a newer version of the same programme? You know then what I am talking about.
- Reusing the ideas/concepts, once found in a document and managed to open that very document, usually involves huge amount of re-formatting. This will produce mistakes. Ever tried to use a BibTeX) generated reference list, found in a PDF file in a new article? I found better ways to spend my nights and weekends (yes, I am married and I have a garden).
- Over time, it can become very difficult to distinguish between different versions of a document, concept and/or idea. As it happens in real life, things move on even in computing and the related sciences. Documents are written for a specific historic context, which might but often will not appear in their abstract (or the name of the folder their are stored in).
- The above issues do apply to figures and presentations as much as to the text part of documents. Reorganising my documents/figures/presentations I did find way too many duplicates. I have used too many graphic software packages in the past 10 years which don't exist anymore, or which do not run on the latest version of my preferred operating system. Some of the figures are only available in some sort of low-resolution bitmap, rendering them useless even for a non-peer-reviewed article today (the original source

got 'lost', in most cases because someone removed the project folder after the project was terminated).

A solution is to create a unified document repository, then use this repository as 'normative source' to create documents for specific purposes while leaving the text blocks, headings, figures, presentations, references, acronyms and all other reusable 'stuff' in the repository for the next document which might (hopefully will) benefit from them. This can (did it for me already) safe a lot of time, demands archiving (of published documents, thus creating a traceable history), helps to keep important information updated (without jeopardising any other work) and prevents losing any 'stuff'.

The repository needs a few rules, a (customisable) structure but beside that only a bit of effort to be maintained. To give an example: while writing the first version of this article (May 11, 2009), I have moved 4 lecture notes, 2 presentations, 1 book chapter, 1 book (in writing), 1 textbook (for students, with 4 chapters) and 4 articles from my 'mess' into my repository. This involved some re-formatting (plus the occasional re-drawing) to bring the original sources into the target formats. At the same time I did develop the rules of my repository, the structure and the (mostly LaTeX) code (and re-wrote/structured/ruled most of them a few times). I ended up with 1,314 files in 87 folders, which create 9 articles, 2 books, 1 textbook, 3 lecture notes and this document (note: the number of articles increased, because I could re-assemble 'stuff' for new uses, spending some five minutes per one new article). I did remove roughly 100 pages of text (take the classic Spring LNCS format and you get the point of the number of characters) and some 40 figures (all duplicates). I did find way too many errors in the original sources (most of which have been created by 're-using' them earlier from even more-original-sources).

3 The Concept: Separate Things

You already got the idea that separation is important, reading about published documents and a normative repository. The basic idea is: think separation – separate as much as you can, but not more. I know that this sounds like a strange idea when the goal is a unified repository, but it is essential. So we separate several concerns (taking a concept of distributed system design). So if we want to facilitate re-usability, we have to:

1. separate content of a document from its structure and
2. separate the different parts of a document.

For the impatient:

1. Separating content from structure means to identify small, coherent blocks of information, i.e. text describing a certain aspect or an example, and put them separated into the repository folder.
2. Separating parts of a document means to separate the part that is important for publishing from the part that is important for the content and put them

into different places (one in the publish folder and the other one in the repository older). It also means to build a separate repository for figures, since they could be used in many different small blocks of information.

3.1 Separate Content from Structure: the Repository Folder

Now, separating the structure from the content first. The structure of a document (we stay with classic text documents like articles, books, etc. for a while) is words in sentences in paragraphs in (sub-) sections in chapters (if its a book, of not only sections)[1]. We collect sentences and paragraphs but separate them from headings. LaTeXis doing that already with the macros for chapters and sections. We go one step further and provide a generic way to identify a heading with the SKB macro `\setheading`. This allows to select the appropriate LaTeXheading level at a later stage having the context of that later stage in mind (i.e. it might be a section in an article but a chapter in a book). Now we create a structure for the resulting files in our repository, adding meaningful names to the directories and files. Obviously the names of these folders should provide some idea about the general characterisation of the files they contain. Example? Well, if you collect information from Standard Defining Organisations (SDOs) the top folder could be named `sdo` and the sub-folders using the respective SDO acronyms, such as `omg` for the OMG and `ieee` for the IEEE and `ietf` for the IETF. We put all this in a folder named repository, making it explicit that here is were we find all our normative content. The following example shows how that looks like.

```
[repository] ..... the Repository Folder
├─ sdo ..... the folder with our SDO files
│  ├─ omg ..... files for OMG 'stuff'
│  │  ├─ corba-idl.tex ..... The CORBA IDL language
│  │  ├─ omg-mda.tex ..... The OMG Model-driven Architecture
│  │  └─ [...]
│  ├─ ieee ..... files for IEEE 'stuff'
│  │  ├─ 802-1.tex ..... The IEEE LAN concept
│  │  ├─ ethernet.tex ..... The Ethernet protocol
│  │  └─ [...]
│  ├─ ietf ..... files for IETF 'stuff'
│  │  ├─ dns.tex ..... The DNS protocol
│  │  ├─ uri.tex ..... The URI specifications
│  │  └─ [...]
│  └─ [...]
└─ [...]
```

[1] One very meticulous person might add 'characters' and mention that there are more ways to think about a document's structure. But that person is not me. The structure I used fits the purpose (as of now), if it doesn't anymore I will look further.

The result: we have a structure of files, containing our 'stuff', integrated in a structure of folders that allows us to find it (the better this structure the more helpful it is, and remember this is a 'personal' repository, so your personal flavour is king).

3.2 Separating different Parts of a Document

The next step is to separate the remaining parts of a document based on their semantics. You are probably doing that already if you maintain a database for bibliographic information and have many of your articles using it. But we can and should do much more than that.

3.2.1 Bibliography, Acronyms and Figures

So the combination of LaTeXand BibTeXalready helps us for this separation. Using the acronym package, we can extend this to acronyms. Looking into the highly common re-use of figures, we should look into this as well. Let's take the organisation of bibliographic information as a template. I store them using BibTeXand process them with the biblatex package (but that is not critical for now). My BibTeXdatabase is in a special folder (we can call it `references` for the moment) and it uses a file structure that helps me to find information. This structure is based on the BibTeXand biblatex classification, i.e. inproceedings, article, book, thesis, standard, etc.

Now, I can do the same for figures: put them into a special folder (we can call it `figures` for the moment), which contains the source of the figures and the generated formats I need for my documents (usually PDF, some PNG). Now I can reference these figures from the repository as well as for other use cases, such as web publishing or presentations[2].

Last not least, the `acronym` package allows for an automatic handling of acronyms, including list of acronyms. It is very similar to BibTeX in that it will only show the acronyms used in a document out of a (potentially large) database.

One might also want to create other specific structures, such as for programming code. Dont' stop yourself, it's easier to combine things later (if the separation is not effective) than to separate things that are hugely integrated into each other. For one of my internal projects, a parser generation environment based on ANTLR, I created a special folder for the EBNF specifications along with railroad diagrams. Now I can use them in my book and my papers.

Now we name the folders for the separated content. This is straight forward, although you might want to use different names (don't worry, the skb supports that). We add to the already created repository folder the things we need for figures (`figures`) and combine acronyms and BibTeXin a folder called `database`,

[2]My figures are exclusively in SVG using inkscape (www.inkscape.org). This has the advantage of a standardised, text-based format with many export options. All my figures are in a single root folder, structured very similar to the document folders created above, but separated from it. This makes re-use of figures very easy.

separating the data from all other content [3]. Now the directory structure looks like this:

```
[root] ..... for instance /doc
├─ database ..... folder for all sorts of data for the repository
│  ├─ latex ..... data in LaTeX, such as our acronyms
│  ├─ bibtex ..... BibTeX database
│  └─ [...]
├─ figures ..... folder for figures, my sources are SVG & PDF
│  └─ [...]
└─ repository ..... folder for the text files
   └─ [...]
```

What did we do so far? We did separate the different parts of our documents. The more clinical you are, the better the result will be. But please note: separate as much as you should, not as you could. If you don't find a reason for separating 'stuff', then don't do it!

3.2.2 Publications and Content

Here is were it might get slightly more complicated than in the first few steps. And you might see already that the reason for that is separation! We didn't finish the separation, we have to go one step further. And that means to separate now the contents (with the references and acronyms and figures) from the reason to publish a document. This last step of separation is more conceptual, being focused on the *why?* and *where?* and *how?* we publish, rather than being focused on the *what?* we publish.

So we do publish for many reasons: articles for research, project proposals, reports, lecture notes, standard documents, annotated presentations, sometimes even books. We publish for a specific purpose, in a specific (soon historic) context, using the requested format (and style sheets) and a particular structure of our document that fits the purpose. That means we organise and structure our content every time according to these constrains. Thus we need a new directory structure for that, since we will not reuse that as often as our 'stuff' itself. Remember, we use the skb macro `\skbheading` for headings, not the classical LaTeXmacros like `\section`, so our files effectively do not contain much information about their place in the structure, only that they claim one [4]. This comes in handy now, since all we have actually to do is to assign a document heading level to every repository file we load. Let's create a folder for the published documents and

[3] Now, the reason for the database folder and it's structure is that the whole SKB contains more databases, all of which reside here. If you want to simply apply this to LaTeXdocuments you might want to use a different strutural approach.

[4] Currently experimental, but soon to be ready, there will be an extension to the `\skbheading` macro that allows a little bit more information to be put in the repository files. For the moment this is captured in the `\skbheadingduc` macro.

call it `published` with a set of sub-folders that help us to understand the general context of the publication. My directory structure could look like this:

```
[your repository root] .... path to your repository, like /dev/documents
├── [...]
├── publish ..................... folder for published documents
│   ├── articles ............................. ...such as articles
│   ├── books ................................ ...or books
│   ├── lecture_notes ........... ...or lecture notes for computer science
│   ├── presentations ................... ...or general presentations
│   └── [...]
└── [...]
```

We could, and it usually makes sense to do so, go one step further in that separation. This time within the documents in the `published` folder. The reason is the structure of LaTeX documents: they do need some commands specific to LaTeX, which we don't necessarily want to mix with the commands that input our content (the files from repository). So it would make sense to have another pair of documents here, one containing all LaTeX commands needed to create a document, and one containing all the commands that include our content. Say we have a few articles for state of the art discussions on *naming*, *object-models* and *protocols*, we could create the following structure for the `article` folder :

```
articles ..................................... our articles
├── naming.tex ................. the file creating an article on naming
├── object-models.tex ....... the file creating an article on object-models
├── protocols.tex ............. the file creating an article on protocols
└── tex ........... a folder containing the tex files that include our content
    ├── naming.tex ............ the file including all content for naming
    ├── object-models.tex ... the file including all content for object-models
    └── protocols.tex ......... the file including all content for protocols
```

Now everything is structured, thus simple again. If we are looking for content, we go to the `repository` directory and the directory names help us to find what we are looking for. If we need a figure, we do the same at the `figures` directory. `acronyms` and `bibtex` help with the respective other content. If we want a specific published document, we simply check the `published` directory and will have a look into a `tex` sub-directory to see what content is include how.

Good news, the separation is finished! What have we done? We have separated the contents from the structure by creating, created a separate directory structure for figures, another one for bibliographic data, one for acronyms and finally a complete directory structure for published documents. Content and title form a

pair, include figure, use acronyms and references and are combined in the published documents. At this point we can start calling it document repository.

3.2.3 The Final Directory Structure

As this is the final and complete root directory of our repository:

```
[your repository root] .... path to your repository, like /dev/documents
├── database ..... folder for all sorts of data for the repository
│   ├── latex ... this is were LaTeXdata will be collected, such as our acronyms
│   └── bibtex ..... folder for all BibTeX reference files
├── figures ..... folder for figures, my sources are SVG & PDF
├── publish ..... folder for published documents
│   ├── articles ..... ...such as articles
│   ├── books ..... ...or books
│   ├── lecture_notes ..... ...or lecture notes for computer science
│   └── presentations ..... ...or general presentations
└── repository ..... folder for the text content
```

4 User Manual

The SKB provides macros that simplify file handling and hide some LaTeX code (i.e. for figures) from the user, thus helping everyone to focus on the actual document one wants to write. There are a few macros, and they can be catagorised as follows.

- Files and Headings: `\skbinput` and `\skbheading` are used to load files and manage the headings of documents, i.e. associating a heading with a level in the document structure (section, subsection, etc.)
- Figures: `\skbfigure` is your Swiss Army knife for loading figures and graphics.
- Path and filenames - these macros provide direct access to SKB-managed paths and filenames.
- Input files - here we have a few macros that load .tex files, figures, slides and slide annotations.
- Emphasising text - some macros that deal with typesetting text in different ways to emphasise that text from the surrounding paragraph.
- List styles - SKB specific environments adding specific behaviour to lists and enumerate environments.
- Listings - basically a few pre-defined styles for using the listing environment.
- PDF Info - some macros that help to set meta information in PDF documents.
- Acronyms - some macros that help to load the acronym database and more.
- BibTeX- one macro that loads bibliographic information.

For the impatient, we start with a few examples. The first one shows how to use the SKB to produce a simple article. The second one exmplains how the documentation for the SKB is created using most of the SKB macros. Then we detail the usage of all the macros, following the above introduced categorisation.

4.1 Getting Started

4.1.1 The SKB Distribution

The SKB distribution that one can download from SourceForge or CTAN (CTAN coming soon) contains the source files for the SKB, the generated classes and styles, the generated documentation and the source files for the user guide. The following example shows the structure and content of the SKB distribution.

```
[start folder]
├─ doc ..................... The generated PDFs and User Guide Sources
│  ├─ [user-guide] ..................... Sources for the User Guide
│  ├─ skb.pdf ..................... The generated Documentation
│  └─ skb-guide.pdf ..................... The User Guide only
├─ run ..................... The generated Class and Style Files
│  ├─ skb.cfg ..................... The global Configuration File
│  ├─ skb.sty ..................... The Style File
│  └─ *.cls ..................... The Class Files
└─ source ..................... The Source files
   ├─ skb.dtx ..................... Documented LaTeXSource File
   ├─ skb.ins ..................... The LaTeXInstaller File
   └─ *.txt ..... Manifest, Licence, Todo List and History as plain Text
```

4.1.2 Installation

There are two ways to install the SKB. The first option is have it automatically installed by your LaTeX distribution using TeXLife or the CTAN archive[5]. The second option is a manual installation doing the following steps:

1. Go to your LaTeX distribution to the folder `tex/latex`.
2. Create a folder `skb`.
3. Copy all files from the directory `run` of this package to the newly created folder `tex/latex/skb`.
4. Update the filename database of your LaTeX distibution. Please see the manual or help files of your LaTeX distribution for details.

If you want copy the source and documentation files as well, then do the following steps. We start with the documentation:

1. Go to your LaTeX distribution to the folder `doc/latex`.

[5]Note: This option is not yet supported, since the SKB has not yet been submitted to CTAN.

2. Create a folder `skb`.
3. Copy all files from the directory `doc` of this package to the newly created folder `doc/latex/skb`.

And now the source files of the SKB:

1. Go to your LaTeX distribution to the folder `source/latex`.
2. Create a folder `skb`.
3. Copy all files and directories from the directory `source` of this package to the newly created folder `source/latex/skb`.

Now you have installed the SKB and you are ready to use it.

4.1.3 Rebuild the SKB from Source

The SKB class and style files as well as the documentation can be rebuild from its sources very easily. The class and style files are part of the documented LaTeXsources in the file `source/skb.dtx` and the LaTeXinstaller (`source/skb.ins`) provides all necessary instructions. Simply follow the steps shown in the first part of the following example. All you have to do then is to copy the files created to your LaTeX distribution.

```
#Rebuild Class and Style files
$cd run; latex ../source/skb.ins
-> creates: skb.cfg, skb.sty, skbarticle.cls, skbbook.cls,
   skbbeamer.cls, skblncsbeamer.cls and skblncsppt.cls

#Rebuild Documentation
$cd doc
$pdflatex ../source/skb.dtx            # repeat twice
$pdflatex user-guide/user-guide        # repeat twice
$pdflatex user-guide/ug-slides-anim    # repeat twice
$pdflatex user-guide/ug-slides-noanim  # repeat twice
$pdflatex user-guide/ug-slides-notes   # repeat twice

# remove all files except the PDFs for cleanup
```

The SKB documentation comes in several different ways. The file `source/skb.dtx` contains the documented source while the files in `doc/user-guide` can be used to generate the User Guide without source documentation, the SKB presentation (animated and not animated) and the lecture notes for the presentation.

When processing the file `source/skb.dtx`, the User Guide will automatically be included in the generated PDF if it is found in either of the directories `source/../doc/user-guide` (when using the SKB original distribution) or `source/../doc/latex/skb/user-guide` (when the SKB is already installed with your LaTeXdistribution).

The second part of the example above shows how to generate the complete SKB documentation. Please note that the sequence is partially important, for instance the file `ug-slides-noanim` must be processed before the file `ug-slides-notes`.

Please note that the SKB documentation is heavily using the SKB macros, so you will need to have the style and class files installed before you can rebuild the documentation.

4.1.4 Confguration: skbconfig

`\skbconfig` There are multiple options to configure the SKB. The following list contains all possible options starting with the least significant. That means that the higher priority settings, which overwrite other settings, will be listed at the bottom.

- Change the file `skb.sty` in your LaTeXdistribution. This might require administrator (root) privileges. This option, while possible, is not recommended.
- Change the file `skb.cfg` in your LaTeXdistribution. This might require administrator (root) privileges. This option is suitable for a system wide configuration, say on your own computer or laptop.
- Create a file `skblocal.cfg` in your personal LaTeXstyle/template directory. This will be the most convenient way to configure the SKB. Note: you might need to refresh the file database of your LaTeXdistribution.
- Use `\skbconfig` in your documents.

If you chose option 1 we assume you know what you are doing. In case you chose options 2-3, you can use the macro `\skbconfig` to do the configuration for you. The macro comes with options for all possible settings of the SKB. Table 1 describes all options and shows their default value. Please note that the SKB can currently not handle inputs from directories outside the root hierarchy. However, one can call `\skbconfig` anytime to change the root directory, but be carefull with potential side effects!.

Table 1: Options for skbconfig

Option	**Description**	**Default**
`root`	Sets the root path of the SKB. Everything that the SKB processes should be located below the root.	`/doc`
`pub`	Sets the path for the published documents.	`publish`
`rep`	Sets the path for the repository documents.	`/repository`
`fig`	Sets the path for figures.	`/figures`
`sli`	Sets the path for the slides.	`/transparencies`
`acr`, `acrfile`	The SKBuses the `acronym` package and these two macros dtail the directory (`acr`) and the file (`acrfile`) where the acronyms can be found.	acr: `database/latex` acrfile: `acronym`
`bib`, `bibfile`	These two macros detail the directory (`bib`) and the main file (`bibfile`) where bibliographic information (BibTeX database) can be found.	bib: `database/bibtex` bibfile: `bibliography`

The macro `\skbconfig` requires one argument, which describes where the configuration has been changed. This is helpful in combination with the macro `\skboptionsused` to trace configuration settings. For instance, in the files `skb.cfg` and `skblocal.cfg` we should use the respective filename. When changing configuration settings elsewhere, some other descriptive text should be useful.

The following code shows the example for `\kbconfig`. The first one is the defailt content of the file `skb.cfg`. It basically sets all possible configuration options to their default value.

```
%default content of skb.cfg
\skbconfig[
    root=/doc,fig=figures,sli=slides
    acr=database/latex,acrfile=acronym,
    bib=database/bibtex,bibfile=bibliograhpy,
    rep=repository,pub=publish
]{skb.cfg}

%using relative path for root and no directory structure
\skbconfig[
    root=./,rep=,pub=,fig=,sli=,
    acr=,acrfile=acronym,
    bib=,bibfile=bibliograhpy
]{myfile.tex}
```

If you want to change the configuration settings for a single document without any directory structure, overwriting all default settings (from `skb.sty`, `skb.cfg` and `skblocal.cfg` and using the current relative path, you can use the second examle shown above.

To trace the configuration settings, you can use `\skboptionsused`. Please see ### for details on this macro.

4.1.5 Confguration: View Options Used

`\skboptionsused` This macro will print out a warning including the currently used configuration information and the change list for each of them. For example, if the configuration for `root` has not been changed the output for `root` will be

`- root [skb.sty]: /doc`

but if the configuration for `fig` has been changed using `\skbconfig` to `graphics` the output for `root` will be

`- fib [skb.sty, skbconfig]: graphics`

This macro is automatically called at the end of processing.

When creating the documentation for the SKB by running `pdflatex skb.dtx`, the following output will be created:

```
Package skb Warning: Options last changed by: skb-presentation
(skb)                Change log:
(skb)                - root = skb.sty, ug-slides-noanim.tex
(skb)                - acr = skb.sty
(skb)                - acrfile = skb.sty
(skb)                - bib = skb.sty
(skb)                - bibfile = skb.sty
(skb)                - rep = skb.sty
(skb)                - pub = skb.sty, ug-slides-noanim.tex
(skb)                - fig = skb.sty
(skb)                - sli = skb.sty, skb-presentation
(skb)                Last set Path/File Options:
```

```
(skb)                - file root = user-guide/
(skb)                - path root = user-guide
(skb)                - file acr = user-guide/database/latex/acronym
(skb)                - file bib = user-guide/database/bibtex/bibliography
(skb)                - path bib = user-guide/database/bibtex
(skb)                - path rep = user-guide/repository/
(skb)                - path pub = user-guide//
(skb)                - path fig = user-guide/figures/
(skb)                - path sli = user-guide/slides/ .
```

The change log shows that all configuration options have been set by `skb.sty` and later by `skb.cfg`. Furthermore, the configuration option `root` has been changed by `skb.dtx`.

4.1.6 Creating a Directory Structure

The real power (and possibly madness) of the SKB comes with the separation of different parts of a document into different directory structures. For the user guide, we assume the following general directory structure .

```
[your repository root] .... path to your repository, like /dev/documents
├─ database ..... folder for all sorts of data for the repository
│  ├─ latex ... this is were LaTeXdata will be collected, such as our acronyms
│  └─ bibtex ..... folder for all BibTeX reference files
├─ figures ..... folder for figures, my sources are SVG & PDF
├─ publish ..... folder for published documents
│  ├─ articles ..... ...such as articles
│  ├─ books ..... ...or books
│  ├─ lecture_notes ..... ...or lecture notes for computer science
│  └─ presentations ..... ...or general presentations
└─ repository ..... folder for the text content
```

To create this structure, go to the directory were you want to put all your documents, say `/doc`. Now create the folders `database`, `figures`, `publish` and `repository` and the respective sub-folders as shown above. Finally, configure the SKB by either editing one of the configuration files or adding the following line to all of your published documents (and of course change the text `myfile.tex` to something that tells you about the location of the configuration change):

```
\skbconfig[root=/doc,
           acr=database/latex,acrfile=acronym,
           bib=database/bibtex,bibfile=bibliograhpy,
           rep=repository,pub=publish,
           fig=figures,sli=slides
          ]{myfile.tex}
```

The directory structures for the publish folder and the repository folder reflect different views to your document base. The publish folder contains documents that are published for a reason (i.e. articles, books, presentations, white papers,

work in progress) while the repository folder contains content, most likely structured following a content-specific categorisation. The choice of how the directory structure looks like is yours, and of course you could also have multiple document directories with completely different structures, for instance one for computer science publications and one for a gardening book. The SKB does not set any limit, since it can be configured very flexibly to your needs (please see subsubsection 4.1.4 for more details) .

4.2 Files, Figures and Slides

4.2.1 Files and Headings

`\skbinput` `\skbheading`

Just to remember: we have two different types of files in two different directory structures. The repository folder stores the content and the publish folder stores everything needed to produce complete documents. For the content in the repository, we mark headings with the macro `\skbheading`. This macro has no options and no arguments and is simply called with the text for the heading, as shown in the following example.

```
\skbheading{My Heading}
```

Leaving the argument empty will have the same effect as calling the original LaTeX macros directly with an empty argument.

The association of a LaTeX document level with the heading is then defined for the published documents (in the publish folder) using the macro `\skbinput`. This macro provides a number of options and requires one argument. The follwing examples shows a few use cases for `\skbinput`. For all possible options, please see Table 3

```
1 \skbinput{myfile}
2 \skbinput[from=rep]{myfile}
3 \skbinput[from=pub]{myfile}
4 \skbinput[level=chapter]{myfile}
5 \skbinput[from=pub,level=chapter]{myfile}
6 \skbinput[from=pub]{test/myfile}
```

Table 3: Options for skbinput

Option	Description	Values
`from`	Set the directory from where the file should be loaded.	`pub`, `rep`, `fig`, `sli`
`level`	Set the document level to be used for the next occurance of `\skbheading`	`book`, `part`, `title`, `chapter`, `section`, `subsection`, `subsubsection`

Let's start with the simpliest form, one argument only shown in line 1. The macro will look for a file called myfile.tex in the root directory of the SKB. The file extension .tex is automatically added, and since we did not specify any special

directory the root directory is used instead. If the file is not found, the macro will throw an error providing the full path and filename it did try to load.

Line 2 shows how we can load the file `myfile.tex` from the repository folder. As you can see, the option `from` is supplied with the argument `rep`, which in fact directs the macro to look for `myfile.tex` in the repository folder. Should the file be located in the folder for published documents, we simply change the `from` option to `pub` as shown in line 3.

If `from` is used and neither `pub` nor `rep` is given, the macro will trow an error.

To associate a document level for the heading, we can use the option `level` to define which level we want. This option understands all standard document levels from the memoir package: book, part, title, chapter, section, subsection and subsubsection. So, if we want to load myfile.tex as a chapter we simple invoke `\skbinput` as shown in line 4 of the example.

Since myfile.tex is part of the repository, we combine the two options `from` and `level` (see line 5). This call to `\input` will load `myfile.tex` from the repository and use `\chapter` for the heading found in that file. If `myfile.tex` is in a sub folder, we simply add that sub folder to the filename. An example is shown in line 6 assuming the the file is located in the repository sub-folder examples.

4.2.2 Figures

`\skbfigure`

The classic way to add figures to your document is to have a PDF or PNG or JPG file ready, include it using `\includegraphics` while putting it into a box to resize it (i.e. to the width of the text in your document), putting this very box into a figure environment so that LaTeX can process list of figures etc. and of course adding lable and caption to it. Here is some LaTeX example, which also uses the center environment:

```
\begin{figure}\begin{center}
    \resizebox{\textwidth}{!}{
      \includegraphics[width=\textwidth]{../figures/myfig}}
  \caption{My Figure}\label{myfig}
\end{center}\end{figure}
```

With the SKB macro `\skbfigure` things become a little bit simplier. takes a number of options and one argument. The following code shows a number of examples for using this macro.

```
1 \skbfigure{myfig}
2 \skbfigure[figure,center]{myfig}
3 \skbfigure[figure,center,width=\textwidth]{myfig}
4 \skbfigure[figure,center,
5           caption=My Figure,label=myfig]{myfig}
```

Let's start with the easy usage, simply using the one argument to load a figure, as shown in line 1. This call will simply use `\includegraphics` and `\resizebox` to load the figure `myfig` from the figure directory, so we do not need to say `../figures` anymore. To use the figure and the center environment, we simply add two options requesting exactly that, as shown in line 2. In other words, using the option `figure` will put the `myfig` in a figure environment and using the option `center` will center the figure.

Similar for width and height information. Say the figure should be rescaled to the width of the text in your document you simply add width to the options, as shown in line 3 Use `height` for height or both options if required. Note that the width and the hight are automatically applied to the `\resizebox` and `\includegraphics`. You can also add caption and label information using the respective options (lines 4 and 5). Now we will have the same result as the classi LaTeX example. You can also add the required position for your figure, if using the figure environment applying the option `position` with the usual paramters, including `H` from the float environment.

Table 4: Options for skbfigure

Option	**Description**
`width`	Set the width to be used with `\esizebox` and `\ncludegraphics`.
`height`	Set the height to be used with `\esizebox` and `\ncludegraphics`.
`center`	Use center environment.
`figure`	Use figure environment.
`position`	The position to be used within figure environment. This option will be ignored if not combined with `figure`.
`caption`	The caption to be used. Ignored if the option `figure` is not used.
`label`	The label to be used. Ignored if the option `figure` is not used.
`multiinclide`	The label to be used. Ignored if the option `figure` is not used.

The last option for the macro `\skbfigure` is called `multiinclude`. It can be used with the beamer package to realise animations by loading a series of images and showing them in sequence with or without overlaying. If used, this option will overwrite all other options resulting in a simple call to `\ultiinclude` within a resised box. One can use all standard multiinclude paramters with `\skbfigure`, just omit the enclosing brackets. For instance, if you want to use multiinclude on the `myfig` with the options `<+->` call

```
\skbfigure[multiinclude=+-]{myfig}
```

The figure size will be automatically set to `\textwidth` and the height to `!`. The start of the multiinclude is fixed to be 0, the format is PDF. For more informatio on how to use multiinclude please refer to mpmulti and beamer packages.

4.2.3 Slides

`\skbslide` This macro helps to create lecture notes (handouts) using PDF slides and LaTeX notes without using the beamer package. The reason for adding this to the SKB was to integrate slides from sources outside the LaTeX universe (i.e. Microsoft Powerpoint). Some of my presentations are done using Powerpoint, but for

handouts I do prefer using LaTeX thus benefiting from many of the automated features it provides (references, acronyms). As a nice side effect, the output generated is scalable (assuming that the PDF sources of the slides contain scalable images rather than bitmaps, which can be easily realised using for instance Inkscape's EMF export within Microsoft Powerpoint slides).

The macro `\skbslide` provides all means to include PDF slides with or without annotations, annotations only and it can load the annotations using different mechanisms. The macro offers two options to set the input path for the slides and the annotations: `slidefrom` and `notefrom`. If `slidefrom` is used, then the slide (PDF) file will be loaded from the requested path (`sli`, `rep` or `pub`). If `notefrom` is used, then the annotation (TEX) file will be loaded from the requested path (`sli`, `rep` or `pub`). The default path for slides and annotations is the path for slides.

The third option `annotate` requests to load annotations. If not used, no annotations will be loaded. It can be used in combination with the two arguments to automated loading annotations.

The two arguments of this macro define the files for the slide and the annotation. They can be used as followes:

- Argument 1 is the slide to be loaded. If a name if given, we load the PDF using `\inputgraphics` with width being `\textwidth`. If no name is given, no slide will be loaded.
- Argument 2 is the file with the annotations in combination with the option `annotate`. If this option is not used then no annotations will be loaded. If the option is used and no name is given, then the annotation is loaded from a file with the same name as the slide plus the extension `.tex`. If this option is used and a name is given then this file will be loaded.

This provides the following combinations for `\skbslide`

- Slide only: argument 1 has the name for the PDF, argument 2 is empty
- Annotation only: argument 1 is empty, argument 2 has the name for the TEX, option `annotate` used
- Slide with Annotation 1: argument 1 has the name for the PDF, argument 2 has the name for the TEX, option `annotate` used
- Slide with Annotation 2: argument 1 has the name for the PDF, argument is empty, option `annotate` used
- do nothing: leave both arguments empty

Some examples on how to use `\skbslide:`

```
1 \skbslide{myslides/slide1}{}
2 \skbslide{myslides/slide2}{}\clearpage
3 \skbslide[annotate]{myslides/slide3}{}
4 \skbslide[annotate,notefrom=rep]
5     {myslides/theme1}{text/theme1}
6 \skbslide[annotate,notefrom=rep,slidefrom=rep]
7     {text/theme2}{text/theme2}
```

In line 1 and 2 we load `myslides/slide1.pdf` and `myslides/slide2.pdf` from the default directory without any annotations and clear the page after that. In

line 3 we load `myslides/slide2.pdf` and request this slide to be annotated without giving a specific file name, thus loading `myslides/slide3.tex`, both files from the default slides directory. In lines 4&5 we change the directory for the notes and request a particular file to be loaded, resulting in the slide loaded as `myslides/theme1.pdf` from the slides directory and the annotations loaded as `text/theme1.tex` from the repository. Finally, in lines 6&7 we change both folders to the repository, this loading `text/theme2.pdf` and `text/theme2.tex` from the repository.

\skbslidecite

The macro `\skbslidecite` provides some simple means to add citations to annotated slides. It takes two arguments, the first one for the type of citation and the second one for the actual citation. Here a simple example:

```
1 \skbslidecite{Slide}{\cite{tanenbaum-andrew:book:2003}}
2 \skbslidecite{Notes}{\cite{standard:IETF:RFC:1155}}
```

The first line states that the slide contains material from a book of Tannenbaum and the second line states that the annotation contains material from an IETF RFC standard documents (RFC 1155). Since this macro is very simple, any content can be given for the two arguments.

4.3 Filenames, Acronyms and References

4.3.1 Path and File Names

\skbfileroot
\skbpathroot
\skbfileacr
\skbfilebib
\skbpathbib
\skbfilerep
\skbfilepub
\skbfilefig
\skbfilesli

The SKB provides a number of macros to directly create path and file names. Most of these macros are actually used within the SKB , but they might also be useful for users to access files without using the provided specialised macros (such as `\skbinput`. The following macros are provided:

- `\skbpathroot` – returns the set root path of the SKB .
- `\skbfileroot` – returns the set root path and adds `/#1`, i.e. the directory separator and the argument provided.
- `\skbfileacr` – returns the path (including root) and file name for the acronym database.
- `\skbfilebib` – returns the path (including root) and file name for the file that loads the reference database (BibTeX).
- `\skbpathbib` – returns the path (including root) to the reference database.
- `\skbfilerep` – returns the path to the repository and adds `/#1`, i.e. the directory separator and the argument provided.
- `\skbfilepub` – returns the path to the folder with the published documents and adds `/#1`, i.e. the directory separator and the argument provided.
- `\skbfilefig` – returns the path to the figure folder and adds `/#1`, i.e. the directory separator and the argument provided.
- `\skbfilesli` – returns the path to the slide folder and adds `/#1`, i.e. the directory separator and the argument provided.

4.3.2 Loading Acronyms and Bibliographic Information

\skbacronyms
\skbbibtex

These two macros can be used to load the acronym database (`\skbacronyms`)

and the references (\skbbibtex). Both macros behave identical: they use \InputIfFileExists to check whether the acronym or bibtex file exists. If so, they simply input the file. If not, they use \PackageError to throw an error with a help message, showing the requested database file to input. One should use \skbacronyms at the place in the document were the list of acronyms should be printed and \skbbibtex at the beginning of the document to load the bibliographic information.

4.4 Other useful Macros

4.4.1 Emphasising Text

\skbem Highlighting or emphasising text is an important aspect of many technical documents. One can use LaTeXmacros directly to set text in italic or bold. This has the disadvantage that there is no meaningful information given as on why that text is treated in a special way. Furthermore, when the editor requires to change certain highlights, it will be very difficult to go through a large document and figure out which text is to be changed.

To prevent that from happening, one can use LaTeXmacros to actually distingnguish between different highlighted text. A simple start is provided by the SKB. It is simply because, at the moment, it only supports three different ways and no furhter meaningful information. But it is a start.

The macro \skbem comes with three different options. The option bold will set the text given in the argument in bold face. Similar, the option italic will set it italic. Last not least, the option code will use another SKB macro (\skbcode) for typesetting the argument text. The following code shows some examples for the macro:

```
Use \cmd{\skbem} to produce \skbem[bold]{bold},
\skbem[italic]{italic} or \skbem[code]{type writer} text.

The example above shows the macro \skbem[code]{skbem} with
the option \skbem[italic]{bold} and \skbem[bold]{italic}.
```

And here the tinal type setting of that example:

Use \skbem to produce **bold**, *italic* or `type writer` text.

The example above shows the macro skbem with the option *bold* and **italic**.

\skbcode This macro \skbcode is a facade for calling the macro \stinline from the listing package with a basic style that uses type writer font (ttfamily).

4.4.2 Environments for lists and enumerates

\skbnotelist \skbnoteenum These two environments mimic the macro \tightlists from the memoir package. It might be usefull when not using memoir to minimise the margin between items in lists (iemize) and enumerations (enumerate).

Both environments do the following:

- Store current value of \parskip and \itemsep.
- Set \parskip and \itemsep to 0cm.

- Use the original environments (itemize for skbnotelist and enumerate for skbnoteenum)
- Set `\parskip` and `\itemsep` back to thir original value.

Here is an example using first the classic list environment (itemize) and then the SKBmacro `\kbnotelist` [6] [7] :

- Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
- Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
- Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
- Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Now list with `\skbnotelist`:

- Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
- Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
- Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
- Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Note: both macros will only change the margins of the memoir package is not loaded!

4.4.3 Listings Styles and Support

The SKB comes with a few predefined styles for the listing package. Most of them use type writer font in scriptsize, arrange a grey box around the listing and set the keywords to Blue4.

- generic – for any generic listing without specifying a language and no line numbers.
- genericLN – same as generic, just with line number in the left side, which means allowing extra space left to the listing box.

[6] For those who are interested, the 'Lorem Ipsum' is the standard phrase commonly used since the 1500s.

[7] The following examples might have no effect on annotated slides...

- gentab – almost the same as generic, but without definitions for frame and numbers, which seem to collide with some table environments.
- genericLNspecial – same as genericLN, just with a lighter grey for the box.
- beamer-example – style designed for examples in beamer frames.
- beamer-exampleLN – same as beamer-example, just with line numbers on the left, which means allowing extra space left to the listing box.
- javaCode – generic style plus lanugage Java.

`\lstdefinestyle` There is also one macro supported, which sets the listing style back to normal, i.e. after changing it in the text. Some macros in the SKB make use of this. All that `\lstdefinestyle` does is setting the basic style back to type writer font.

4.4.4 Optional Text – Versions and Optional

The SKB provides two means to include text and other LaTeXcommands on an otional basis. They are pre-configured and will be automatically set/unset according to the three main document types the SKB supports:

- text – is equivalent to any classic text document, for instance an article or a book.
- slide – is used to idenify slides, for instance beamer frames.
- note – is used to identify lecture notes or handouts, in essence annotated slides (frames).
- anim – for beamer frames, used for text with animation activated.
- noanim – for beamer frames, used for text with animation deactivated.
- memoir – used for documents that include the memoir package.

We use the packages versions and optional and support both. The main difference is that with versions one has to use `\beging` and `\end` while with optional one can use more than one of the above introduced types. The macros for provided for optional text are:

- `\skbmodetext` and options using `text` – will be valid if neither beamer nor beamerarticle is loaded (normal text).
- `\skbmodeslide` and options using `slide` – will be valid if the beamer package is loaded (slides).
- `\skbmodenote` and options using `note` – will be valid if the beamerarticle package is loaded (annotated slides).
- `\skbmodeanim` and options using `anim` – will be valid if the beamer package is loaded and the SKB is loaded with the argument `beameranim`
- `\skbmodenoanim` and options using `noanim` – will be valid if the beamer package is loaded and the SKB is loaded with the argument `beamernoanim`
- `\skbmodememoir` and options using `memoir` – will be valid if the memoir package is loaded

The following code shows a few examples on how to use the optional text.

5 Examples

A Simple Article

Take the article that describes the state of the art in protocols. Remember, we have all the contents for that in our `repository` directory. We go the directory that has the published articles `published/articles` and create a new file say `protocols.tex` .

```
\documentclass{skbarticle}

\begin{document}
  \author{Sven van der Meer}
  \title{Protocols, Formats and Communication Services}
  \maketitle
  \tableofcontents*
  \bigskip

  \skbinput[from=rep]{sota/protocols}

  \section{Introduction}
    \skbinput[from=rep,level=subsection]
        {sota/protocols/data_encoding}
    \skbinput[from=rep,level=subsection]
        {sota/protocols/message-formates}
    \skbinput[from=rep,level=subsection]
        {sota/protocols/protocols}
    \skbinput[from=rep,level=subsection]
        {sota/protocols/protocol-services}

  \skbinput[from=rep,level=section]{sdo/omg/corba-giop}
  \skbinput[from=rep,level=section]{sdo/ietf/snmp-protocol}
  \skbinput[from=rep,level=section]{sdo/itu/x700-cmip}
  \skbinput[from=rep,level=section]{sdo/w3c/http}

\end{document}
\endinput
```

The article uses the class `skbarticle`. That class will load the SKB package and the memoir class and do all settings we need. It prepares the title page and prints the table of contents like any other LaTeXarticle. The it uses `\skbinput` to load files from the repository. The first one is loaded without requesting a level. In other words, there is some text right at the beginning of our article, without any special heading, like an abstract.

Then we do start the section 'Introduction' and collect a few files with their heading categorised as sub-sections. Reading the directory and file names, we can already guess what the introduction will be doing: it introduces general protocol concepts with regard to data encoding, protocol message formats, protocols themselves and protocol services. The last block loads four files with headings categorised as sections. Using the directory names, we see that the remaining article describes the protocols GIOP defined by the OMG, SNMP by the IETF, CMIP by the ITU-T and finally HTTP by the W3C.

Finally, we load acronyms and bebliography and finishing the article. This example will create a table of contents similar to this:

```
1 Introduction . . . . . 1
```

Job done. Now we can use LaTeX or PDF-LaTeX to compile our article.

6 Implementation: Kernel

First we do announce the package.

```
⟨*skbpackage⟩
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{skb}[2010/08/04 Sven's Knowledge Base - SKB for LaTeX v0.5]
```

Next we process the package's options. To do that, we define a new if that indicates if we process slides with or without animation, and then we set that new if accordingly.

```
\newif\if@skbBeamerAnim
\@skbBeamerAnimfalse
\DeclareOption{beameranim}{\@skbBeamerAnimtrue}
\DeclareOption{beamernoanim}{\@skbBeamerAnimfalse}
\ProcessOptions\relax
```

6.1 Required Packages

Now we load a few packages that we need within the SKB. We use keyval to allow for options in macros, the listings package for all listings, dirtree to show tree structures similar to a directory tree, ifpdf to establish whether we use PDF or not, datetime to get the current date and the versions package to allow for optional text. Note: some packages, such as the package optional, are loaded at a later stage.

```
\RequirePackage{keyval}
\RequirePackage{listings}
\RequirePackage{dirtree}
\RequirePackage{ifpdf}
\RequirePackage{datetime}
\RequirePackage{versions}
```

6.2 Conditiona/Optional Text Support

Now we set everything that we need to provide optional text. Basically, we want to distinguish between the following modes: text (normal text), slide (for slides),

note (for slite annotations), anim (for animated slides, noanim (for non-animated slides) and memoir (if we use the memoir package).

We start with the memoir package. First we define a configuration value (used when loading the package optional) and a new if (telling us later if memoir is loaded or not).

```
\def\skb@cfg@memoir{}
\newif\ifSkbMemoirLoaded
```

Now we test for the memoir package. Note, if this package is loaded after the SKB, this test and all following actions will fail. If the package is loaded, then we set the if to true, activate (include) the environment skbmodememoir and set our configuration value to the string ", memoir". If the memoir package is not loaded, then we set the if to false, deactivate (exclude) the environment skbmodememoir and load the package booktabs (to provide the commands `\toprule` and `\bottomrule`.

```
\@ifclassloaded{memoir}
  {\SkbMemoirLoadedtrue
   \includeversion{skbmodememoir}
   \def\skb@cfg@memoir{,memoir}}
  {\SkbMemoirLoadedfalse
   \excludeversion{skbmodememoir}
   \RequirePackage{booktabs}}
```

Now we check for the style beamerarticle. We define an if, set its default value to false and test for of the package is loaded (if so, we change the if to true).

```
\newif\ifSkbBeamerArticleLoaded
\SkbBeamerArticleLoadedfalse
\@ifpackageloaded{beamerarticle}{\SkbBeamerArticleLoadedtrue}{}
```

Now we check for the beamer package. e define an if, set its default value to false and test for of the package is loaded (if so, we change the if to true).

```
\newif\ifSkbBeamerLoaded
\SkbBeamerLoadedfalse
\@ifclassloaded{beamer}{\SkbBeamerLoadedtrue}{}
```

Now we process the first optional text support. First, we define a configuration value for beamer animations. If animations are requested (skb package option, see above), we set that value to the string ",anim" and activate (include) the environment skbmodeanim and deactivate (exclude) the environment skbmodenoanim. If no-animation is requested (skb package option, see above) or as default we set the value to the string ",noanim" and deactivate (exclude) the environment skbmodeanim and activate (include) the environment skbmodenoanim.

```
\def\skb@cfg@beameranim{}
\if@skbBeamerAnim
  \def\skb@cfg@beameranim{,anim}
  \excludeversion{skbmodenoanim}
```

```
  \includeversion{skbmodeanim}
\else
  \def\skb@cfg@beameranim{,noanim}
  \excludeversion{skbmodeanim}
  \includeversion{skbmodenoanim}
\fi
```

Now we are ready to provide for all other optional text support. The code configures the environments skbmodetext, skbmodenote and skbmodeslide and loads the optional package depending if we have the beamer package loaded or have the package beamerarticle loaded or have none of the two packages loaded. The environments (package versions) are excluded or included accordingly. The package optional is loaded with the respective option activated (text, note or slide) and using the two configuration values we have defined above (these values are either empty having no effect or contain the option to be included).

```
\ifSkbBeamerLoaded
  \excludeversion{skbmodetext}
  \excludeversion{skbmodenote}
  \includeversion{skbmodeslide}
  \RequirePackage[slide\skb@cfg@memoir\skb@cfg@beameranim]{optional}
\else\ifSkbBeamerArticleLoaded
  \excludeversion{skbmodetext}
  \includeversion{skbmodenote}
  \excludeversion{skbmodeslide}
  \RequirePackage[note\skb@cfg@memoir\skb@cfg@beameranim]{optional}
\else
  \includeversion{skbmodetext}
  \excludeversion{skbmodenote}
  \excludeversion{skbmodeslide}
  \RequirePackage[text\skb@cfg@memoir\skb@cfg@beameranim]{optional}
\fi\fi
```

6.3 Provide Command

\SKB
\BibTeX
\DescribeMacro
\cmdprint
\cmd

The SKB provides for a few commands that the documentation (and maybe your documents as well) expect to be available. The first two are for typesetting SKB and BibTeX, the rest are simply usefull.

```
\providecommand{\SKB}{{\scshape SKB}}
\providecommand{\BibTeX}{{\scshape Bib}\TeX}
\providecommand{\DescribeMacro}[1]{\relax}
\providecommand{\cmdprint}[1]{\texttt{\string#1}}
\providecommand{\cmd}[1]{\cmdprint{#1}}%
```

6.4 Macro Redefinitions

The SKB documentation uses the package dirtree and we want to have some of its default settings changed. For the comments, the default configuration we want is an small, italic serif font in blue; and for the style part we want a type writer font in black.

```
\renewcommand*\DTstylecomment{\itshape\sffamily\color{blue}\small}
\renewcommand*\DTstyle{\ttfamily\textcolor{black}}
```

6.5 At End of Document

Last not least, we define what should happen at the end of the processing of the input document. At them moment, we call `\skbpdfinfo` to set PDF meta information and `\skboptionsused` to print out the change log and current set of SKB configuration options.

```
\AtEndDocument{
  \skbpdfinfo
  \skboptionsused
}
```

6.6 Package Configuration

The basic idea of the SKB is that different parts of a document (figures, slides, repository, published documents) reside in different folders. So the main configuration of the SKBis to provide macros to set and get these folders and to load files from them.

To simplify coding, we introduce some macros that handle configuration information. These macros will be used by the SKB package to define, set and get configuration information. The macros also store the origin of changes to the configuration information.

`\skb@tmp` This variable is used to temporarily store macros and strings. The value can change anytime a new SKB macro is called.

```
\newcommand{\skb@tmp}{}
```

`\skb@cfg@origlast` Is used to store the last location (second argument of `\skbconfig`) were any configuration information has been changed. The currently possible locations are `skb.sty` for default values, `skb.cfg` for the general configuration file, `skblocal.cfg` for the local configuration file and `skbconfig` when the macro `\skbconfig` was called.

```
\newcommand{\skb@cfg@origlast}{skb.sty}
```

`\skb@defCfgVars` This macro is used to define new configuration information. It defines two new macros, one for the name of the configuration information and one for storing a

change log. The first argument is the name to be used and the second argument the default initialisation. For instance, to add the configuration information for the root path with the default value '/doc' call

```
\skb@defCfgVars{root}{/doc}
```

```
\newcommand{\skb@defCfgVars}[2]{
  \@namedef{skb@cfg@var@#1}{#2}
  \@namedef{skb@cfg@orig@#1}{skb.sty}
}
```

`\skb@setCfgVars` Alter configuration information and append the location from where its called (second argument of `\skbconfig` taken from `\kb@cfg@origlast`) to the change log.

```
\newcommand{\skb@setCfgVars}[2]{
  \@namedef{skb@cfg@var@#1}{#2}
  \expandafter\protected@edef\csname skb@cfg@orig@#1\endcsname%
    {\csname skb@cfg@orig@#1\endcsname,\space \skb@cfg@origlast}%
}
```

`\skb@getCfgVars` This macro provides access to configuration values. It is used everywhere in the SKB to retrieve configuration values.

```
\newcommand{\skb@getCfgVars}[1]{%
  \csname skb@cfg@var@#1\endcsname%
}%
```

Now we use `\skb@defCfgVars` to initialise all configuration values the SKB uses.

`\skb@cfg@var@root` The first one is the root directory. Everything that the SKB processes should be located below the root. The SKB can currently not handle inputs from directories outside the root hierarchy (Note: one can call `\skbconfig` anytime to change the root directory, but be carefull with potential side effects!). The default value for the root directory is `/doc`.

```
\skb@defCfgVars{root}{/doc}
```

`\skb@cfg@var@acr` `\skb@cfg@var@acrfile` These two values define the directory and the file name for the acronym database. The SKB uses the `acronym` package and the two macros detail the directory (`acr`) and the file (`acrfile`) where the acronyms can be found. The default for the directory is `database/latex` and the default for the file is `acronym`.

```
\skb@defCfgVars{acr}{database/latex}
\skb@defCfgVars{acrfile}{acronym}
```

`\skb@cfg@var@bib` `\skb@cfg@var@bibfile` These two values define the directory and the file name for the BibTeXdatabase. The two macros detail the directory (`bib`) and the main file (`bibfile`) where bibliographic information can be found. The default for the directory is `database/bibtex` and the default for the file is `bibliography.tex`.

```
\skb@defCfgVars{bib}{database/bibtex}
```

```
\skb@defCfgVars{bibfile}{bibliography}
```

`\skb@cfg@var@rep` This value points to the `repository` directory. The default value is `repository`.

```
\skb@defCfgVars{rep}{repository}
```

`\skb@cfg@var@pub` This value points to the folder with the published documents. The default value is `publish`.

```
\skb@defCfgVars{pub}{publish}
```

`\skb@cfg@var@fig` This value points to the directory for figures. The default value is `figures`.

```
\skb@defCfgVars{fig}{figures}
```

`\skb@cfg@var@sli` This value points to the directory for slides. The default value is `transparencies`.

```
\skb@defCfgVars{sli}{transparencies}
```

6.7 Generic Input Macro

`\skb@input@doife` `\skb@input@doife` is the generic input macro. It expects four arguments. The first argument is the SKB macro that should be used to input a file. The second argument is the actual file to be loaded, without file extension. The third argument is the file extension to be used. The fours argument is plain text that should be added to the help message in case an arror occured while loading the file. If the second and third argument are empty, we assume that the first argument already contains directory and file and file extension information.

```
\newcommand{\skb@input@doife}[4]{%
  \def\filearg{#2}
  \ifx\filearg\empty%
    \edef\intfile{\csname #1\endcsname}%
  \else%
    \edef\intfile{\csname #1\endcsname{#2}#3}%
  \fi%
  \InputIfFileExists{\intfile}{}%
    {\PackageError{skb}%
      {file not found: \intfile}%
      {I did not find the requested file #4,%
       \MessageBreak please check: \intfile%
       \MessageBreak <return> to continue, no file loaded}%
    }%
}
```

6.8 Kernel support for skbinput

This is the actual core functionality of the SKB package: flexibly load files from various pre-defined locations (folders). We start with a few macros that we can use later to test options using the package keyval.

`\skb@input@var@rep` This macro represents the string "rep", which will be later used to test for macro options, for instance in `\skbinput`.

```
\def\skb@input@var@rep{rep}
```

`\skb@input@var@pub` This macro represents the string "pub", which will be later used to test for macro options, for instance in `\skbinput`.

```
\def\skb@input@var@pub{pub}
```

`\skb@input@var@fig` This macro represents the string "fig", which will be later used to test for macro options, for instance in `\skbinput`.

```
\def\skb@input@var@fig{fig}
```

`\skb@input@var@sli` This macro represents the string "sli", which will be later used to test for macro options, for instance in `\skbinput`.

```
\def\skb@input@var@sli{sli}
```

The next set of macros will load files from various supported folders. All of them behave identical: they expect argument 1 being the reuqest file and use `\InputIfFileExists` to check whether this file exists. If so, they simply input the file using `\input`. If not, they use `\PackageError` to throw an error with a help message, showing the requested directory and file. The extention .tex is automatically added to the argument, which in turn should only contain the path and the basename of the file.

`\skb@input@doroot` Load a given .tex file from the root directory.

```
\newcommand{\skb@input@doroot}[1]{%
  \def\intarg{#1}
  \skb@input@doife{skbfileroot}{\intarg}{.tex}{in given location}
}
```

`\skb@input@dorep` Load a given .tex file from the repository.

```
\newcommand{\skb@input@dorep}[1]{%
  \def\intarg{#1}
  \skb@input@doife{skbfilerep}{\intarg}{.tex}{in the repository}
}
```

`\skb@input@dopub` Load a given .tex file from the directory with the published documents.

```
\newcommand{\skb@input@dopub}[1]{%
  \def\intarg{#1}
  \skb@input@doife{skbfilepub}{\intarg}{.tex}{in the published document folder}
}
```

`\skb@input@dofig` Load a given .tex file from the figure directory.

```
\newcommand{\skb@input@dofig}[1]{%
  \def\intarg{#1}
  \skb@input@doife{skbfilefig}{\intarg}{.tex}{in the figure folder}
```

```
}
```

`\skb@input@dosli` Load a given .tex file from the slide directory.

```
\newcommand{\skb@input@dosli}[1]{%
  \def\intarg{#1}
  \skb@input@doife{skbfilesli}{\intarg}{.tex}{in the slide folder}
}
```

`\skb@input@call` `\skb@input@set` These two macros are used to load files. `\skb@input@call` will point to the currently requested load macro (see above). `\skb@input@set` sets the default load option in `\skb@input@call` to `\skb@input@doroot`. That means if no option is given for an input directory, then the SKB root directory will be used.

```
\def\skb@input@call{}
\newcommand\skb@input@set{%
  \gdef\skb@input@call{\skb@input@doroot}
}
```

7 Implementation: Configuring the SKB

7.1 Changing Configuration: skbconfig

7.1.1 The Macro Options

The macro provides one option per SKB configuration value. Each option expects one paramter; the new value. The options are `root` (for the root directory), `acr` (for the acronym directory), `acrfile` (for the acronym file), `bib` (for the bibtex directory), `bibfile` (for the bibtex file), `rep` (for the repository directory), `pub` (for the directory with the published documents) and `sli` (for the directory with slides).

```
\define@key{skbconfig}{root}[]{\skb@setCfgVars{root}{#1}}
\define@key{skbconfig}{acr}[]{\skb@setCfgVars{acr}{#1}}
\define@key{skbconfig}{acrfile}[]{\skb@setCfgVars{acrfile}{#1}}
\define@key{skbconfig}{bib}[]{\skb@setCfgVars{bib}{#1}}
\define@key{skbconfig}{bibfile}[]{\skb@setCfgVars{bibfile}{#1}}
\define@key{skbconfig}{rep}[]{\skb@setCfgVars{rep}{#1}}
\define@key{skbconfig}{pub}[]{\skb@setCfgVars{pub}{#1}}
\define@key{skbconfig}{fig}[]{\skb@setCfgVars{fig}{#1}}
\define@key{skbconfig}{sli}[]{\skb@setCfgVars{sli}{#1}}
```

7.1.2 The Macro

`\skbconfig` This macro allows to change the main directory and path information for the SKB. It reads the provided options and changes the requested values in the SKB. The

macro takes one argument which will set the origin of the configuration change. If this argument is empty, the origin will be set to `skbconfig`.

```
\newcommand{\skbconfig}[2][]{
  \def\intarg{#2}
```

If no second argument is given, then set `\skb@cfg@origlast` to the string "skbconfig" (this macro's name) otherwise use the second argument to set `\skb@cfg@origlast`. In both cases, print out a general warning about the change of configuration values for later trace or debugging.

```
  \ifx\intarg\empty
    \renewcommand{\skb@cfg@origlast}{skbconfig}
    \PackageWarning{skb}{load options overwritten by skbconfig}
  \else
    \renewcommand{\skb@cfg@origlast}{#2}
    \PackageWarning{skb}{load options overwritten by #2}
  \fi
```

Now use the keyval package to process the options. They will set the respective configuration values, so there is nothing else to do here.

```
  \setkeys{skbconfig}{#1}
}
```

7.2 Changing Configuration: skb.cfg and akblocal.cfg

The SKBcan also be configured using external configuration files. Two files will be loaded if they exist:

- `skb.cfg` – Should be used with the installed package in your TEX/LATEX distribution. If it exists, it will overwrite the default options for directories and paths.
- `skblocal.cfg` – Should be used in your local styles/template directory. If it exsits, it will overwrite the default options as well as the options loaded with `skb.cfg`.

We use `\InputIfFileExists` to test if the configuration file exist. If true, we load the configuration file and print out a general warning for later trace or debugging. If not, we simply do nothing.

```
\InputIfFileExists{skb.cfg}{%
  \PackageWarning{skb}{load options from skb.cfg}
}{}
\InputIfFileExists{skblocal.cfg}{%
  \PackageWarning{skb}{load options from skblocal.cfg}
}{}
```

7.3 Viewing Configuration: skboptionsused

`\skboptionsused` This macro can be used to print out a message (as package warning), which contains the change log and the currently used value for all SKB configuration values.

```
\newcommand{\skboptionsused}{
  \PackageWarningNoLine{skb}{%
    Options last changed by: \skb@cfg@origlast \MessageBreak
    Change log: \MessageBreak
    - root = \skb@cfg@orig@root \MessageBreak
    - acr = \skb@cfg@orig@acr \MessageBreak
    - acrfile = \skb@cfg@orig@acrfile \MessageBreak
    - bib = \skb@cfg@orig@bib \MessageBreak
    - bibfile = \skb@cfg@orig@bibfile \MessageBreak
    - rep = \skb@cfg@orig@rep \MessageBreak
    - pub = \skb@cfg@orig@pub \MessageBreak
    - fig = \skb@cfg@orig@fig \MessageBreak
    - sli = \skb@cfg@orig@sli \MessageBreak
    Last set Path/File Options: \MessageBreak
    - file root = \skbfileroot{} \MessageBreak
    - path root = \skbpathroot \MessageBreak
    - file acr = \skbfileacr \MessageBreak
    - file bib = \skbfilebib \MessageBreak
    - path bib = \skbpathbib \MessageBreak
    - path rep = \skbfilerep{} \MessageBreak
    - path pub = \skbfilepub{} \MessageBreak
    - path fig = \skbfilefig{} \MessageBreak
    - path sli = \skbfilesli{}
  }
}
```

8 Implementation: Files, Figures and Slides

8.1 Declaring Headings: skbheading

`\skbheading` This macro can be used everywhere to declare a new heading and let the SKB decide which document level to use. The actual document level must be declared in the loading file using `\skbinput` with the option `level`, otherwise this command will have no effect.

```
\newcommand{\skbheading}[1]{
  \ifx\empty\skb@inputLevel
    #1
  \else%
    \skb@inputLevel{#1}%
  \fi
}
```

8.2 Loading TEX files: skbinput

8.2.1 Macro Options

skbinput: opt from

The option `from` is used to point to one of the following SKB directories: the repository (`from=rep`), the folder with the published documents (`from=pub`), the figure folder (`from=fig`) or the slide folder (`from=sli`). The option is optional, but when used must give one of the those values. The SKB will throw an error otherwise. The implementation works as follows: if the option is used, its paramter is evaluated. Depending on which SKB directories is requested, the value `\skb@input@call` is set to point to the respective load load macro. For instance, if the requested directory is the repository (`from=rep`) then `\skb@input@call` will be pointed to `\skb@input@dorep`.

```
\define@key{skbinput}{from}[]{%
  \def\intarg{#1}
  \ifx\skb@input@var@rep\intarg
    \gdef\skb@input@call{\skb@input@dorep}
  \else\ifx\skb@input@var@pub\intarg
    \gdef\skb@input@call{\skb@input@dopub}
  \else\ifx\skb@input@var@fig\intarg
    \gdef\skb@input@call{\skb@input@dofig}
  \else\ifx\skb@input@var@sli\intarg
    \gdef\skb@input@call{\skb@input@dosli}
  \else
    \PackageError{skb}%
      {Value for option \@tempa\space not supported: \intarg}%
      {I do not know the value \intarg\space for the option \@tempa.%
       \MessageBreak Please use either "rep", "pub", "fig" or "sli".%
       \MessageBreak <return> to continue, no file will be loaded}
  \fi\fi\fi\fi
}
```

skbinput: opt level

The option `level` is used to define the document level to be used for the next occurance of `\skbheading`. Supported are all document levels known to LaTeX and no check is done whether the currently used document class supports them or not (for instance, the article class does not support the document level chapter, however, memoir supports it even in article mode). The supported paramters for this option are: `book` (memoir pacakge), `part` (memoir pacakge), `title` (base LaTeX classes), `chapter` (LaTeX book class), `section` (base LaTeX classes), `subsection` (base LaTeX classes) and `subsubsection` (base LaTeX classes).

The option is optional, but when used must give one of the above described values. The package will throw an error otherwise.

We start be defining the macros we use later for testing the option. This might be a slightly awkward way to do it, I am still looking into optimising this code. Anyway, we define everything we need for book, part, title, chapter, section, subsection and subsubsection.

```
\def\skb@inputLevelBook{book}
\def\skb@inputLevelPart{part}
\def\skb@inputLevelTitle{title}
\def\skb@inputLevelChapter{chapter}
\def\skb@inputLevelSection{section}
\def\skb@inputLevelSubSection{subsection}
\def\skb@inputLevelSubSubSection{subsubsection}
```

Now we define a macro that will be used to point to the selected input level (`\skb@inputLevel`) and a macro that will be used to set the default input level to be empty (i.e. do nothing, `\skb@SetInputLevel`).

```
\def\skb@inputLevel{}
\newcommand\skb@SetInputLevel{\gdef\skb@inputLevel{}}
```

And here is the actual definition of the option `level`. For each supported parameter (introduced and defined above) we test if it was provided calling the option (put into `\\intarg` on start) and if so we point `\skb@inputLevel` to the LaTeX macro realising that document level. For instance, if the requested level is subsection we point `\skb@inputLevel` to the LaTeX macro `\subsection`. That means we can later simply call `\skb@inputLevel` to instruct LaTeX to realise the requested document level. In case the parameter is not supported, the option will throw an error along with a help message.

```
\define@key{skbinput}{level}[]{%
  \def\intarg{#1}
  \ifx\skb@inputLevelBook\intarg
    \let\skb@inputLevel=\book
  \else\ifx\skb@inputLevelPart\intarg
    \let\skb@inputLevel=\part
  \else\ifx\skb@inputLevelTitle\intarg
    \let\skb@inputLevel=\title
  \else\ifx\skb@inputLevelChapter\intarg
    \let\skb@inputLevel=\chapter
  \else\ifx\skb@inputLevelSection\intarg
    \let\skb@inputLevel=\section
  \else\ifx\skb@inputLevelSubSection\intarg
    \let\skb@inputLevel=\subsection
  \else\ifx\skb@inputLevelSubSubSection\intarg
    \let\skb@inputLevel=\subsubsection
  \else
    \PackageError{skb}%
      {Value for option \@tempa\space not supported: \intarg}%
      {I do not know the value \intarg\space for the option \@tempa.%
       \MessageBreak Please use only: book, part, title, chapter,%
       \MessageBreak section, subsection or subsubsection.%
       \MessageBreak <return> to continue, no level will be set and heading is ignored}
  \fi\fi\fi\fi\fi\fi\fi
}
```

8.2.2 The Macro

\skbinput This macro will load a .tex file from the root directory or from an SKB known directory (if option `from` is applied). It will also configure the document level macro for the next use of `\skbjeading`, if the option `level` is applied. If `level` is not used, then `\skbheading` will have no effect. The macro first sets the input level to be empty (`\skb@input@set`) and the input macro to the default value (`\skb@input@set`). The it processes the options (using the keyval pacakge) and finally calls `\skb@input@call` to realise the load of the requested file.

```
\newcommand\skbinput[2][]{%
  \skb@input@set
  \skb@SetInputLevel
  \setkeys{skbinput}{#1}
  \skb@input@call{#2}
}
```

8.3 Loading Figures: skbfigure

8.3.1 Macro Options

This macro supportes a number of options. To be able to test for the applied options, we first define a few macros that will be used by `\skbfigure` to realise the requested figure input. We define one macro per option supported.

```
\def\skb@FigureOptWidth{}
\def\skb@FigureOptHeight{}
\def\skb@FigureOptCenter{}
\def\skb@FigureOptFigure{}
\def\skb@FigureOptPosition{}
\def\skb@FigureOptCaption{}
\def\skb@FigureOptLabel{}
\def\skb@FigureOptMultiinclide{}
```

To be able to reset all of these macros before processing a figure, we define a reset macro.

```
\newcommand{\skb@figureOptReset}{
  \gdef\skb@FigureOptWidth{}
  \gdef\skb@FigureOptHeight{}
  \gdef\skb@FigureOptCenter{}
  \gdef\skb@FigureOptFigure{}
  \gdef\skb@FigureOptPosition{}
  \gdef\skb@FigureOptCaption{}
  \gdef\skb@FigureOptLabel{}
  \gdef\skb@FigureOptMultiinclide{}
}
```

Now we define all options for `\skbfigure`. All options work the same way: they either take the parameter given and put it into the corresponding macro we defined above or simply set the corresponding macro to true. This way we can test these corresponding macros for being empty (default) or not and then decide how to process the figure input.

`skbfigure opt width` The first one is called `width` used for the width of `\resizebox` and `\ncludegraphics`.

```
\define@key{skbfigures}{width}[]{%
  \gdef\skb@FigureOptWidth{#1}
}
```

`skbfigure opt height` The option height is used for the height of `\resizebox` and `\ncludegraphics`.

```
\define@key{skbfigures}{height}[]{%
  \gdef\skb@FigureOptHeight{#1}
}
```

`skbfigure opt center` The option center is used to trigger the center environment (so it only needs to set true).

```
\define@key{skbfigures}{center}[true]{%
  \gdef\skb@FigureOptCenter{true}
}
```

`skbfigure opt figure` The option figure is used to trigger the figure environment (so it only needs to set true).

```
\define@key{skbfigures}{figure}[true]{%
  \gdef\skb@FigureOptFigure{true}
}
```

`skbfigure opt position` The option position is used to fix the position when figure environment is used

```
\define@key{skbfigures}{position}[]{%
  \gdef\skb@FigureOptPosition{\begin{figure}[#1]}
}
```

`skbfigure opt caption` The option caption is used to define the caption of the figure used as `\caption`

```
\define@key{skbfigures}{caption}[]{%
  \gdef\skb@FigureOptCaption{\caption{#1}}
}
```

`skbfigure opt label` The option label is used to define the label of the figure used as `\label`

```
\define@key{skbfigures}{label}[]{%
  \gdef\skb@FigureOptLabel{\label{fig:#1}}
}
```

`skbfigure opt multiinclude` The option multiinclude is a special option to use `\multiinclude`, automatically deactivates all other options

```
\define@key{skbfigures}{multiinclude}[]{%
```

```
  \gdef\skb@FigureOptMultiinclide{#1}
}
```

8.3.2 The Macro

\skbfigure `\skbfigure` itself expects options (processed using keyval) and the actual file to be included. The file name should start at the figure root directory.

```
\newcommand{\skbfigure}[2][]{
```

First, we call our reset function and then use keyval to process the options.

```
  \skb@figureOptReset
  \setkeys{skbfigures}{#1}%

```

Now we process the options figure and position to decide if and how to use the figure environment. If the figure option has been used, we test if the position option has been used as well. If figure and position have been used, we call `\skb@FigureOptPosition`, which expands to `\beginfigure[option]`. If only the figure option was used, we directly invoke `\beginfigure`.

```
  \ifx\skb@FigureOptFigure\empty\else
    \ifx\skb@FigureOptPosition\empty
      \begin{figure}
    \else
      \skb@FigureOptPosition
    \fi
  \fi
```

Next is the center option. If it was used, we call `\begincenter`.

```
  \ifx\skb@FigureOptCenter\empty\else\begin{center}\fi

```

The core of the macro. If the option multiinclude was not used, we proceed load the figure as we would usually do with LaTeX. If multiinclude was used, then we simply call `\multiinclude` with the given overlay information, starting at number 0, using PDF format and scaling everything to `\textwidth`.

```
  \ifx\skb@FigureOptMultiinclide\empty
    \ifx\skb@FigureOptWidth\empty
      \ifx\skb@FigureOptHeight\empty
        \resizebox{!}{!}%
        {\includegraphics[]%
        {\skbfilefig{#2}}}
      \else
        \resizebox{!}{\skb@FigureOptHeight}%
        {\includegraphics[height=\skb@FigureOptHeight]%
        {\skbfilefig{#2}}}
      \fi
```

```
    \else
      \ifx\skb@FigureOptHeight\empty
        \resizebox{\skb@FigureOptWidth}{!}%
        {\includegraphics[width=\skb@FigureOptWidth]%
        {\skbfilefig{#2}}}
      \else
        \resizebox{\skb@FigureOptWidth}%
                  {\skb@FigureOptHeight}%
        {\includegraphics[%
            width=\skb@FigureOptWidth,%
            height=\skb@FigureOptHeight%]%
        {\skbfilefig{#2}}}
      \fi
    \fi
  \else
    \resizebox{\textwidth}{!}%
    {\multiinclude[<\skb@FigureOptMultiinclide>]%
    [start=0,format=pdf,graphics={width=\textwidth}]%
    {\skbfilefig{#2}}}
  \fi

```

If we did use the figure environment, then we check for given caption and label.

```
  \ifx\skb@FigureOptFigure\empty\else%
    \skb@FigureOptCaption
    \skb@FigureOptLabel
  \fi%

```

And finally we close the figure and center environments if we did open them earlier.

```
  \ifx\skb@FigureOptCenter\empty\else\end{center}\fi
  \ifx\skb@FigureOptFigure\empty\else\end{figure}\fi
}
```

8.4 Loading Slides: skbslide

This macro allows to load a (configurable) combination of PDF slide and LaTeX annotation to be loaded in a single call.

8.4.1 Some Extentions

`\skb@slides@callpath` The first is a macro that will maintain the current path and file for loading slides.

```
\def\skb@slides@callpath{}
```

`\skb@slides@doslinote` The second is a macro to load annotations from the slide folder.

```
\newcommand{\skb@slides@doslinote}[1]{%
```

```
  \def\intarg{#1}
  \skb@input@doife{skbfilesli}{\intarg}{.tex}{in the slides folder}
}
```

8.4.2 Macro Options

`\skbslideopt slidefrom`

The option `slidefrom` is used to point to one of the following SKB directories: `sli` (the folder for slides) or `pub` (the folder for published documents) or `rep` (the repository directory). The option is optional, but when used must give one of the above described values. The SKB will throw an error otherwise.

```
\define@key{skbslide}{slidefrom}[]{%
  \def\intarg{#1}
  \ifx\skb@input@var@sli\intarg
    \let\skb@slides@callpath=\skbfilesli
  \else\ifx\skb@input@var@pub\intarg
    \let\skb@slides@callpath=\skbfilepub
  \else\ifx\skb@input@var@rep\intarg
    \let\skb@slides@callpath=\skbfilerep
  \else
    \PackageError{skb}%
      {Value for option \@tempa\space not supported: \intarg}%
      {I do not know the value \intarg\space for the option \@tempa.%
       \MessageBreak Please use either "pub", "rep" or "sli".%
       \MessageBreak <return> to continue, no file will be loaded}
  \fi\fi\fi
}
```

`\skbslideopt notefrom`

The option `notefrom` is used to point to one of the following SKB directories: `sli` (the folder for slides) or `pub` (the folder for published documents) or `rep` (the repository directory). The option is optional, but when used must give one of the above described values. The SKB will throw an error otherwise.

```
\define@key{skbslide}{notefrom}[]{%
  \def\intarg{#1}
  \ifx\skb@input@var@sli\intarg
    \gdef\skb@input@call{\skb@slides@doslinote}
  \else\ifx\skb@input@var@pub\intarg
    \gdef\skb@input@call{\skb@input@dopub}
  \else\ifx\skb@input@var@rep\intarg
    \gdef\skb@input@call{\skb@input@dorep}
  \else
    \PackageError{skb}%
      {Value for option \@tempa\space not supported: \intarg}%
      {I do not know the value \intarg\space for the option \@tempa.%
       \MessageBreak Please use either "pub", "rep" or "sli".%
       \MessageBreak <return> to continue, no file will be loaded}
  \fi\fi\fi
}
```

\skbslideopt annotate The option annotate requests to load annotations for the slide. If not given, no annotations will be loaded.

```
\def\skb@slides@loadnote{}
\define@key{skbslide}{annotate}[true]{%
  \gdef\skb@slides@loadnote{true}
}
```

8.4.3 The Macro

\skbslide This macro will load the slide and annotation, depending on the options provided.

```
\newcommand\skbslide[3][]{%
  \gdef\skb@slides@loadnote{}
  \gdef\skb@input@call{\skb@slides@doslinote}
  \let\skb@slides@callpath=\skbfilesli
  \setkeys{skbslide}{#1}

  \def\sl{#2}
  \def\an{#3}

  \ifx\sl\empty\else
    \begin{figure}[!bh]
      \resizebox{\textwidth}{!}{\includegraphics[width=\textwidth]{\skb@slides@callpath{#2}}}
    \end{figure}
  \fi

  \ifx\skb@slides@loadnote\empty\else
    \ifx\an\empty
      \skb@input@call{#2}
      \clearpage
    \else
      \skb@input@call{#3}
      \clearpage
    \fi
  \fi
}
```

\skbslide This simple macro can help to provide standardised citations on annotation pages.

```
\newcommand{\skbslidecite}[2]{\small Source \textit{#2}: \textit{#1} \normalsize}
```

9 Implementation: Filenames, Acronyms and References

9.1 Path and File Names

These macros are used within the SKB to generate path and filenames for all known directories and files. They basically provide user-level access to kernel-level processed configuration date. All path names, except root, are fully qualified from root. All filenames are fully qualified from root. Macros that expect an argument use that very argument as the reuqested filename to provide path and filename.

`\skbpathroot` This macro returns the currently set root path.

```
411 \newcommand{\skbpathroot}{\skb@getCfgVars{root}}
```

`\skbfileroot` This macro takes the given argument and prefixes the root path to it.

```
412 \newcommand{\skbfileroot}[1]{\skb@getCfgVars{root}/#1}
```

`\skbfileacr` This macro returns the file of the acronym database.

```
413 \newcommand{\skbfileacr}{\skb@getCfgVars{root}/\skb@getCfgVars{acr}/\skb@getCfgVars{acrfile}}
```

`\skbpathbib` This macro returns the path to the reference library.

```
414 \newcommand{\skbpathbib}{\skb@getCfgVars{root}/\skb@getCfgVars{bib}}
```

`\skbfilebib` This macro returns the file that is used to load the reference library.

```
415 \newcommand{\skbfilebib}{\skb@getCfgVars{root}/\skb@getCfgVars{bib}/\skb@getCfgVars{bibfile}}
```

`\skbfilerep` This macro takes the provided argument and prefixes the path to the repository to it.

```
416 \newcommand{\skbfilerep}[1]{\skb@getCfgVars{root}/\skb@getCfgVars{rep}/#1}
```

`\skbfilepub` This macro takes the provided argument and prefixes the path to the published documents to it.

```
417 \newcommand{\skbfilepub}[1]{\skb@getCfgVars{root}/\skb@getCfgVars{pub}/#1}
```

`\skbfilefig` This macro takes the provided argument and prefixes the path to the figures to it.

```
418 \newcommand{\skbfilefig}[1]{\skb@getCfgVars{root}/\skb@getCfgVars{fig}/#1}
```

`\skbfilesli` This macro takes the provided argument and prefixes the path to the slides to it.

```
419 \newcommand{\skbfilesli}[1]{\skb@getCfgVars{root}/\skb@getCfgVars{sli}/#1}
```

9.2 Loading Acronyms

`\skbacronyms` This macro will load the acronym database. It should be used at the place in your

document were you want the list of acronyms to appear. If the file is not found, an error is thrown.

```
\newcommand{\skbacronyms}{%
  \skb@input@doife{skbfileacr}{}{}{for acronym database}
}
```

9.3 Loading Reference Database

`\skbbibtex` This macro will load the reference database. It should be used before you start the actual document. If the file is not found, an error is thrown.

```
\newcommand{\skbbibtex}{%
  \skb@input@doife{skbfilebib}{}{}{for bibtex database}
}
```

10 Implementation: Other useful Macros

10.1 Emphasising Text: skbem

10.1.1 Macro Options

`skbem opt italic` This option will typset the given text for `\skbem` using italic font.

```
\def\skb@emCmd{}
\define@key{skbem}{italic}[true]{%
  \gdef\skb@emCmd{\textit}%
}%
```

`skbem opt bold` This option will typset the given text for `\skbem` using bold font.

```
\define@key{skbem}{bold}[true]{%
  \gdef\skb@emCmd{\textbf}%
}%
```

`skbem opt code` This option will typset the given text for `\skbem` using the command `\skbcode` (see below).

```
\define@key{skbem}{code}[true]{%
  \gdef\skb@emCmd{\skbcode}%
}%
```

10.1.2 The Macro

`\skbem` This macro helps to emphasise text in an explicit way (as compared to use font commands within the actual text). Simply call with the one of the option to emphasise text.

```
\newcommand{\skbem}[2][]{%
  \gdef\skb@emCmd{}%
  \setkeys{skbem}{#1}%
  \skb@emCmd{#2}%
}%
```

10.2 Emphasising Text: skbcode

`\skbcode` This macro is a facade for calling `\lstinline` with basicstyle set to type writer font. It is used by `skbem` with the option `code` to call `\lstinline` but can also be called directly.

```
\newcommand{\skbcode}[1]{%
  \lstinline[basicstyle=\ttfamily]{#1}%
}%
```

10.3 List Environments: skbnotelist and skbnoteenum

These environments simulate `\tightlist` from the memoir package. They work identical: call the environment itemize (for skbnotelist) or enumerate (for skbnoteenum), temoprarily store the values of `\parskip` and `\temsep`, set the two values to 0 (thus minimising the margin between items) and on exit simply restore these two values

`\skb@TmpParskp` `\skb@TmpItemsep` These two variables temporarily store `\parskip` and `\itemsep`.

```
\def\skb@TmpParskp{}
\def\skb@TmpItemsep{}
```

`\skbnotelist` New Environment skbnotelist to minimise the margin between list items.

```
\newenvironment{skbnotelist}
  {
    \begin{itemize}%
    \ifSkbMemoirLoaded\else
      \gdef\skb@TmpParskp{\parskip}\setlength{\parskip}{0cm}%
      \gdef\skb@TmpItemsep{\itemsep}\setlength{\itemsep}{0cm}%
    \fi
  }
  {
    \end{itemize}%
    \ifSkbMemoirLoaded\else
      \setlength{\parskip}{\skb@TmpParskp}%
      \setlength{\itemsep}{\skb@TmpItemsep}%
    \fi%
  }
```

`\skbnoteenum` New Environment skbnotelist to minimise the margin between list items.

```
\newenvironment{skbnoteenum}%
  {
    \begin{enumerate}%
    \ifSkbMemoirLoaded\else
      \gdef\skb@TmpParskp{\parskip}\setlength{\parskip}{0cm}%
      \gdef\skb@TmpItemsep{\itemsep}\setlength{\itemsep}{0cm}%
    \fi
  }
  {
    \end{enumerate}%
    \ifSkbMemoirLoaded\else
      \setlength{\parskip}{\skb@TmpParskp}%
      \setlength{\itemsep}{\skb@TmpItemsep}%
    \fi%
  }
```

10.4 Acronyms in Footnotes: skbacft

`\skbacft` This macro provides some functionality that the `acronym` package does not offer: introducing acronyms in a footnote (if they are used the first time) or simply use the short form. I found this is useful when writing books, where sometimes introducing acronym in the normal text flow somehow disturbs that very flow.

```
\newcommand{\skbacft}[1]{%
  \ifAC@dua
     \ifAC@starred\acl*{#1}\else\acl{#1}\fi%
  \else
  \expandafter\ifx\csname ac@#1\endcsname\AC@used%
    \acs{#1}%
  \else
    \acs{#1}\footnote{\acf{#1}}%
    \fi
  \fi}
```

10.5 PDF Meta Information: skbpdfinfo and more

`\skbtitle` This macro allows to set text for the title of the generated PDF.

```
\def\skb@TitleText{}
\newcommand{\skbtitle}[1]{\gdef\skb@TitleText{#1}}
```

`\skbauthor` This macro allows to set text for the author of the generated PDF.

```
\def\skb@AuthorText{}
\newcommand{\skbauthor}[1]{\gdef\skb@AuthorText{#1}}
```

`\skbsubject` This macro allows to set text for the subject of the generated PDF.

```
\def\skb@SubjectText{}
\newcommand{\skbsubject}[1]{\gdef\skb@SubjectText{#1}}
```

`\skbkeywords` This macro allows to set text for the keywords of the generated PDF.

```
\def\skb@KeywordsText{}
\newcommand{\skbkeywords}[1]{\gdef\skb@KeywordsText{#1}}
```

`\skbpdfinfo` This macro will set the PDF information in the generated PDF. It first checks if we are in PDF mode, and then uses the information from `\skb@AuthorText`, `\skb@TitleText` plus subject and keywords from above.

```
\newcommand{\skbpdfinfo}{%
  \ifpdf
    \pdfinfo{
       /Author (\skb@AuthorText)
       /Title  (\skb@TitleText)
       /ModDate (D:\pdfdate)
       /Subject (\skb@SubjectText)
       /Keywords (\skb@KeywordsText)
    }
  \fi
}
```

10.6 Listing Styles and Support

The SKB comes with a few pre-defined styles for the `listing` package. Most of these predefined styles use type writer font in scriptsize, arrange a grey box around the listing and set the keywords to Blue4.

The first style is the for any generic listing without specifying a language and no line numbers.

```
\lstdefinestyle{generic}
     {basicstyle=\scriptsize\ttfamily, backgroundcolor=\color[gray]{.9},
      frame=single, framerule=.5pt, numbers=none,
      linewidth=0.99\textwidth, xleftmargin=3pt,
      keywordstyle=\bfseries\color{Blue4},
      identifierstyle=\bfseries}
```

This style is designed for listings within tables. It is similar to the generic one above, except that the definitions for frame and numbers are not used, which seem to collide with some table environments.

```
\lstdefinestyle{gentab}
     {basicstyle=\scriptsize\ttfamily, backgroundcolor=\color[gray]{.9},
      framerule=0pt,
      linewidth=.86\textwidth, xleftmargin=3pt,
      keywordstyle=\bfseries\color{Blue4},
      identifierstyle=\bfseries}
```

This style is the same as the generic one above, except that it switches on line numbers and allows extra space for them within the grey box.

```
\lstdefinestyle{genericLN}
    {basicstyle=\scriptsize\ttfamily, backgroundcolor=\color[gray]{.9},
     frame=single, framerule=.5pt, numbers=left,
     linewidth=0.99\textwidth, xleftmargin=20pt,
     keywordstyle=\bfseries\color{Blue4},
     identifierstyle=\bfseries}
```

This style is based on the style `genricLN`, basically using a slightly brighter grey for the box.

```
\lstdefinestyle{genericLNspecial}
    {basicstyle=\small\ttfamily, backgroundcolor=\color[gray]{.97},
     frame=single, framerule=.5pt, numbers=left,
     linewidth=0.99\textwidth, xleftmargin=20pt,
     keywordstyle=\bfseries\color{Blue4},
     identifierstyle=\bfseries}
```

This style is designed for examples within slides (frames) using the `beamer` package.

```
\lstdefinestyle{beamer-example}
    {basicstyle=\scriptsize\ttfamily,
     frame=single, framerule=0pt, numbers=none,
     linewidth=0.99\textwidth, xleftmargin=3pt,
     keywordstyle=\bfseries\color{Blue4},
     identifierstyle=\bfseries}
```

This style is designed for examples within slides (frames) using the `beamer` with added line numbers.

```
\lstdefinestyle{beamer-exampleLN}
    {basicstyle=\scriptsize\ttfamily,
     frame=single, framerule=0pt, numbers=left,
     linewidth=0.99\textwidth, xleftmargin=20pt,
     keywordstyle=\bfseries\color{Blue4},
     identifierstyle=\bfseries}
```

This style uses the definitions from the generic style above and set the language to Java.

```
\lstdefinestyle{javaCode}
    {basicstyle=\scriptsize\ttfamily, backgroundcolor=\color[gray]{.9},
     frame=single, framerule=0pt, language=JAVA,
     numbers=none,
     keywordstyle=\bfseries\color{Blue4},
     identifierstyle=,
     linewidth=0.99\columnwidth}
```

This style can be used to set 'normal' style after changing it.

```
\lstdefinestyle{inText}
    {basicstyle=\ttfamily}
```

11 Experimental Macros

This part of the SKB is experimental. Please do not use it for production code or important documents. The macros in this section will be moved as soon as they are stable, or simply removed. They can, as long as they stay in this section, be changed at any time in future releases.

11.1 Defining new relative Headings: skbheadingudc

When we set the document level with `\skbheading`, it might be usefull to actually have a macro that allows to relatively change headings. This is usefull if we have more than one heading in a repository file, where the first one defines the heading and will get an associative document level from the calling document while any subsequent heading might need to go one level up or down. The macro here works as long as we don't need to recursively store document levels. So it is not stable right now and makes only sense if used for single headings.

First, a macro that we use to point to the new heading (rather than the one used by `\skbinput`.

```
\def\skb@newHeading{}
```

11.1.1 Macro Options

Now the option down, which indicates that this heading should be one level down from the previous one.

```
\define@key{skbheadings}{down}[true]{%
  \ifx\skb@inputLevel\part
    \let\skb@newHeading=\chapter
    \let\skb@inputLevel=\chapter
  \else\ifx\skb@inputLevel\chapter
    \let\skb@newHeading=\section
    \let\skb@inputLevel=\section
  \else\ifx\skb@inputLevel\section
    \let\skb@newHeading=\subsection
    \let\skb@inputLevel=\subsection
  \else\ifx\skb@inputLevel\subsection
    \let\skb@newHeading=\subsubsection
    \let\skb@inputLevel=\subsubsection
  \else
    \KV@err{Invalid current level for SkbNewHeading(down),
            please use: part, chapter, section or subsection}
   \fi\fi\fi\fi
}
```

Now the option up, which indicates that this heading should be one level up from the previous one.

```
\define@key{skbheadings}{up}[true]{%
  \ifx\skb@inputLevel\chapter
    \let\skb@newHeading=\part
    \let\skb@inputLevel=\part
  \else\ifx\skb@inputLevel\section
    \let\skb@newHeading=\chapter
    \let\skb@inputLevel=\chapter
  \else\ifx\skb@inputLevel\subsection
    \let\skb@newHeading=\section
    \let\skb@inputLevel=\section
  \else\ifx\skb@inputLevel\subsubsection
    \let\skb@newHeading=\subsection
    \let\skb@inputLevel=\subsection
  \else
    \KV@err{Invalid current level for SkbNewHeading(up),
            please use: chapter, section, subsection or subsubsection}
   \fi\fi\fi\fi
}
```

Now the option last, which indicates that this heading should be on the same level as the previous one.

```
\define@key{skbheadings}{last}[true]{%
  \let\skb@newHeading=\skb@inputLevel%
}
```

11.1.2 The Macro

`\skbheadingudc`

```
\newcommand{\skbheadingudc}[2][]{%
  \gdef\skb@newHeading{}
  \setkeys{skbheadings}{#1}%
  \ifx\empty\skb@newHeading\else%
    \skb@newHeading{#2}%
  \fi
}
```

⟨/skbpackage⟩

12 The Configuration File skb.cfg

This file is used to overwrite the default values for the SKB configuration options. It calles the macro `\skbconfig` using all possible options of that very macro and

providing usefull text as origin of the configuration change `skb.cfg`. Use this as template for the local configuration file `skblocal.cfg` if you need one.

```
⟨*skbcfg⟩
\skbconfig[root=/doc,
           acr=database/latex,
           acrfile=acronym,
           bib=database/bibtex,
           bibfile=bibliograhpy,
           rep=repository,
           pub=publish,
           fig=figures,
           sli=slides
          ]{skb.cfg}
⟨/skbcfg⟩
```

13 The SKB Classes

13.1 The Class skbarticle

This class is an example on how to use the SKB with memoir. I use skbarticle for my articles. Using this class as a template, one can easily write other classes or change/overwrite the settings done here.

First, we announce the package and the font definition file.

```
⟨*skbarticle⟩
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{skbarticle}[2010/08/04 The SKB Article class v0.5]
```

Now we load the memoir class with the following options:

- 10pt - for 10 point font size
- a4paper - I am European, so A4 paper makes sense here
- extrafontsizes - tbd
- twoside - I want my articles to be set with different even/odd pages
- onecolumn - I don't necessarily like 2-columns for my articles
- openright - tbd
- article - use memoir as if it is an article

```
\LoadClass[10pt,a4paper,extrafontsizes,twoside,onecolumn,openright,article]{memoir}
```

Load the SKB.

```
\RequirePackage{skb}
```

13.1.1 Loaded Packages

I prefer BibLaTeX over plain BibTeX, and other parts of the SKB (such as the LAMP server) produce BibLaTeX. The options are:

- style - is set to alphanumeric, much better to find/remember references. If writing for IEEE or LNCS, numeric would be the prefered option.
- sorting - is set to none, not needed here.
- hyperref - I want to have hyperef with my citations

```
615 \RequirePackage[style=alphabetic,sorting=none,hyperref]{biblatex}
```

Load the acronym package and print only the acronyms actually used in the document. This might move into the SKB package later

```
616 \RequirePackage[printonlyused]{acronym}
```

Load a view packages that I tend to use quite often:

- etoolbox - etoolbox
- comment - Add comments to your LaTeX files
- graphicx - Enhanced graphic support, with key/value interface for include graphics
- longtable - Helps with tables that span multiple pages
- colortbl - Allows coloured cells in tables

```
617 \RequirePackage{etoolbox,comment,graphicx,longtable,colortbl}
```

And some more packages needed quite often:

- textcomp - Special characters, such as ® and ©
- gensymb - Generic characters (math and text mode), such as °, ℃, ‰, µ and Ω
- wasysym - Adds characters from *wasy* font, such as ☺, ⊠ and ♲
- units - Typeset units correctly (and produce 'nice' fractions), such as $10\,{}^{\mathrm{m}}/_{\mathrm{s}}$ and ½
- float - Improves interface for floating environments (such as figures, tables)
- xmpmulti - tbd

```
618 \RequirePackage{textcomp,gensymb,wasysym,units,xmpmulti,float}
```

The xcolor package provides driver independent access to all sorts of colour tins, shades, tones and mixes. I like x11names, as you can tell.

```
619 \RequirePackage[x11names]{xcolor}
```

The hyperref package provides layout for hyper references, such as URLs and references within a document, such as acronyms, citations and the table of contents. We use the option colorlings and then provide the colors we prefer for links (linkcolor), citations (citecolor) and URLs (urlcolor).

```
\RequirePackage[colorlinks,%
                linkcolor=AntiqueWhite4,%
                citecolor=SeaGreen4,%
                urlcolor=RoyalBlue3%
               ]{hyperref}
%\RequirePackage[colorlinks,linkcolor=blue]{hyperref}
```

13.1.2 Memoir Options

Not sure, but I don't think semi-iso-pages are good. So not used right now.

```
%\semiisopage
```

Change the margins for even and odd pages. Odd to 1cm and even to 1cm.

```
\setlength{\oddsidemargin}{1cm}
\setlength{\evensidemargin}{0cm}
```

Set width and height for the text. At the moment only the width, to 15cm

```
\setlength{\textwidth}{15cm}
%\setlength{\textheight}{24cm}
```

Don't use chapter numbers in sections, thus making them looking like sections in a classic article (1 instead of the default 0.1)

```
\def\thesection{\arabic{section}}
```

Allow table of contents to go up to sub-sections

```
\settocdepth{subsection}
```

And numbering up to subsubsections

```
\setsecnumdepth{subsubsection}
```

For lists, memoir provides different layouts. We use tightlists here, but can switch that to firmlists if needed

```
\tightlists
%\firmlists
```

What are these for? I forgot...

```
\midsloppy
\raggedbottom
```

13.1.3 Misc Settings

Load the bibliographic information using the SKB.

```
\skbbibtex
```

Finally, we do set the sort option for the bibliography to anyt (biblatex)

```
\ExecuteBibliographyOptions{sorting=anyt}
```

There is no code for `\AtBeginDocument` and `\AtEndDocument`, so we are done now.

⟨/skbarticle⟩

13.2 The Class skbbook

This class is an example on how to use the SKB with memoir. I use skbbook for my books. Using this class as a template, one can easily write other classes or change/overwrite the settings done here.

First, we announce the package and the font definition file.

```
⟨*skbbook⟩
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{skbbook}[2010/08/04 The SKB Book class v0.5]
```

Now we load the memoir class with the following options:

- 11pt - for 11 point font size
- a4paper - I am European, so A4 paper makes sense here
- extrafontsizes - tbd
- twoside - I want my articles to be set with different even/odd pages
- onecolumn - I don't necessarily like 2-columns for my articles
- openright - tbd

```
\LoadClass[11pt,a4paper,extrafontsizes,twoside,onecolumn,openright]{memoir}
```

Load the SKB.

```
\RequirePackage{skb}
```

13.2.1 Loaded Packages

I prefer BibLaTeX over plain BibTeX, and other parts of the SKB (such as the LAMP server) produce BibLaTeX. The options are:

- style - is set to alphanumeric, much better to find/remember references. If writing for IEEE or LNCS, numeric would be the prefered option.
- sorting - is set to none, not needed here.
- hyperref - I want to have hyperef with my citations

```
\RequirePackage[style=alphabetic,sorting=none,hyperref]{biblatex}
```

Load the acronym package and print only the acronyms actually used in the document. This might move into the SKB package later

```
\RequirePackage[printonlyused]{acronym}
```

Load a view packages that I tend to use quite often:

- etoolbox - etoolbox
- comment - Add comments to your LaTeX files
- graphicx - Enhanced graphic support, with key/value interface for include graphics
- longtable - Helps with tables that span multiple pages
- colortbl - Allows coloured cells in tables

```
\RequirePackage{etoolbox,comment,graphicx,longtable,colortbl}
```

And some more packages needed quite often:

- textcomp - Special characters, such as ® and ©
- gensymb - Generic characters (math and text mode), such as °, ℃, ‰, µ and Ω
- wasysym - Adds characters from *wasy* font, such as ☺, ⊠ and ♲
- units - Typeset units correctly (and produce 'nice' fractions), such as $10\,{}^{\mathrm{m}}/_{\mathrm{s}}$ and $^1/_2$
- float - Improves interface for floating environments (such as figures, tables)
- xmpmulti - tbd

```
\RequirePackage{textcomp,gensymb,wasysym,units,xmpmulti,float}
```

The xcolor package provides driver independent access to all sorts of colour tins, shades, tones and mixes. I like x11names, as you can tell.

```
\RequirePackage[x11names]{xcolor}
```

The hyperref package provides layout for hyper references, such as URLs and references within a document, such as acronyms, citations and the table of contents. We use the option colorlings and then provide the colors we prefer for links (linkcolor), citations (citecolor) and URLs (urlcolor).

```
\RequirePackage[colorlinks,%
                linkcolor=AntiqueWhite4,%
                citecolor=SeaGreen4,%
                urlcolor=RoyalBlue3%
               ]{hyperref}
%\RequirePackage[colorlinks,linkcolor=blue]{hyperref}
```

13.2.2 Memoir Options

Not sure, but I don't think semi-iso-pages are good. So not used right now.

```
%\semiisopage
```

Set the head styles to komalike (the other nice style is memman).

```
\headstyles{komalike}
```

Change the margins for even and odd pages. Odd to .5cm and even to 0cm.

```
\setlength{\oddsidemargin}{.5cm}
\setlength{\evensidemargin}{0cm}
```

Set width and height for the text. Width to 15cm and length t0 22cm.

```
\setlength{\textwidth}{15cm}
\setlength{\textheight}{22cm}
```

Get half a centimeter back from the topmargin.

```
\setlength{\topmargin}{-.5cm}
```

Allow table of contents to go up to subsub-sections

```
\settocdepth{subsubsection}
```

And numbering up to subsubsections

```
\setsecnumdepth{subsubsection}
```

For lists, memoir provides different layouts. We use tightlists here, but can switch that to firmlists if needed

```
\tightlists
%\firmlists
```

What are these for? I forgot...

```
\midsloppy
\raggedbottom
```

Chapters shoud look like the memoir veelo style.

```
\chapterstyle{veelo}
```

13.2.3 Misc Settings

Load the bibliographic information using the SKB.

```
\skbbibtex
```

Finally, we do set the sort option for the bibliography to anyt (biblatex)

```
\ExecuteBibliographyOptions{sorting=anyt}
```

There is no code for `\AtBeginDocument` and `\AtEndDocument`, so we are done now.

```
⟨/skbbook⟩
```

13.3 The Class skbbeamer

This class is an example on how to use the SKB with memoir. I use skbbeamer for my beamer presentations. Using this class as a template, one can easily write other classes or change/overwrite the settings done here.

First, we announce the package and the font definition file and process the options.

```
674 ⟨*skbbeamer⟩
675 \NeedsTeXFormat{LaTeX2e}
676 \ProvidesClass{skbbeamer}[2010/08/04 The SKB Beamer class v0.5]
677 \DeclareOption{beameranim}{\PassOptionsToPackage{\CurrentOption}{skb}}
678 \DeclareOption{beamernoanim}{\PassOptionsToPackage{\CurrentOption}{skb}}
679 \ProcessOptions\relax
```

Now we load the xcolor package and then the beamer class. That should load the x11names some of the SKB listing styles use while not creating any clash between the packages beamer and xcolor.

```
680 \RequirePackage[x11names]{xcolor}
681 \LoadClass[x11names]{beamer}
```

Load the SKB.

```
682 \RequirePackage{skb}
```

13.3.1 Loaded Packages

I prefer BibLaTeX over plain BibTeX, and other parts of the SKB (such as the LAMP server) produce BibLaTeX. The options are:

- style - is set to alphanumeric, much better to find/remember references. If writing for IEEE or LNCS, numeric would be the prefered option.
- sorting - is set to none, not needed here.
- hyperref - I want to have hyperef with my citations

```
683 \RequirePackage[style=alphabetic,sorting=none,hyperref]{biblatex}
```

Load the acronym package and print only the acronyms actually used in the document. This might move into the SKB package later

```
684 \RequirePackage[printonlyused]{acronym}
```

Load a view packages that I tend to use quite often:

- etoolbox - etoolbox
- comment - Add comments to your LaTeX files
- graphicx - Enhanced graphic support, with key/value interface for include graphics
- longtable - Helps with tables that span multiple pages

- colortbl - Allows coloured cells in tables

```
\RequirePackage{etoolbox,comment,graphicx,longtable,colortbl}
```

And some more packages needed quite often:

- textcomp - Special characters, such as ® and ©
- gensymb - Generic characters (math and text mode), such as °, ℃, ‰, µ and Ω
- wasysym - Adds characters from *wasy* font, such as ☺, ⊠ and ☋
- units - Typeset units correctly (and produce 'nice' fractions), such as $10\,{}^{\mathrm{m}}/_{\mathrm{s}}$ and $^1/_2$
- float - Improves interface for floating environments (such as figures, tables)
- xmpmulti - tbd

```
\RequirePackage{textcomp,gensymb,wasysym,units,xmpmulti,float}
```

13.3.2 Misc Settings

Load the bibliographic information using the SKB.

```
\skbbibtex
```

And some default settings for the dirtree package.

```
\renewcommand*\DTstylecomment{\itshape\sffamily\color{blue}\scriptsize}
\setlength{\DTbaselineskip}{10pt}
\DTsetlength{0.2em}{1em}{0.2em}{0.4pt}{1.6pt}
\renewcommand*\DTstyle{\scriptsize\ttfamily\textcolor{black}}
```

There is no code for `\AtBeginDocument` and `\AtEndDocument`, so we are done now.

```
⟨/skbbeamer⟩
```

13.4 The Class skblncsbeamer

This class is an example on how to use the SKB with memoir. I use skblncsbeamer for my beamer based handouts. Using this class as a template, one can easily write other classes or change/overwrite the settings done here.

First, we announce the package and the font definition file.

```
⟨*skblncsbeamer⟩
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{skblncsbeamer}[2010/08/04 The SKB LNCS Beamer class v0.5]
```

Just in case there is no `\titlepage` declared, the beamerarticle wants that.

```
\providecommand{\titlepage}{}
```

Now we load the memoir class with the following options:

- 9pt - for 9 point font size
- a4paper - I am European, so A4 paper makes sense here
- extrafontsizes - tbd
- twoside - I want my articles to be set with different even/odd pages
- onecolumn - I don't necessarily like 2-columns for my articles
- openright - tbd
- article - use memoir as if it is an article
- x11names - this option will be forwarded to the xcolor/graphics packages

```
\LoadClass[9pt,a4paper,extrafontsizes,twoside,onecolumn,openright,article,x11names]{memoir}
```

For Beamer handouts, we need the beamerarticle package to load the frame thumbnails.

```
\RequirePackage{beamerarticle,pgf}
```

Load the SKB.

```
\RequirePackage{skb}
```

13.4.1 Loaded Packages

I prefer BibLaTeX over plain BibTeX, and other parts of the SKB (such as the LAMP server) produce BibLaTeX. The options are:

- style - is set to alphanumeric, much better to find/remember references. If writing for IEEE or LNCS, numeric would be the prefered option.
- sorting - is set to none, not needed here.
- hyperref - I want to have hyperef with my citations

```
\RequirePackage[style=alphabetic,sorting=none,hyperref]{biblatex}
```

Load the acronym package and print only the acronyms actually used in the document. This might move into the SKB package later

```
\RequirePackage[printonlyused]{acronym}
```

Load a view packages that I tend to use quite often:

- etoolbox - etoolbox
- comment - Add comments to your LaTeX files
- graphicx - Enhanced graphic support, with key/value interface for include graphics
- longtable - Helps with tables that span multiple pages
- colortbl - Allows coloured cells in tables

```
\RequirePackage{etoolbox,comment,graphicx,longtable,colortbl}
```

And some more packages needed quite often:

- textcomp - Special characters, such as ® and ©
- gensymb - Generic characters (math and text mode), such as °, ℃, ‰, µ and Ω
- wasysym - Adds characters from *wasy* font, such as ☺, ⊠ and ↻
- units - Typeset units correctly (and produce 'nice' fractions), such as $10\,{}^{\mathrm{m}}/_{\mathrm{s}}$ and $^{1}/_{2}$
- float - Improves interface for floating environments (such as figures, tables)
- xmpmulti - tbd

```
\RequirePackage{textcomp,gensymb,wasysym,units,xmpmulti}
```

13.4.2 Memoir Options

Not sure, but I don't think semi-iso-pages are good. So not used right now.

```
%\semiisopage
```

We do want to list files.

```
\listfiles
```

Change the margins for even and odd pages. Odd to 0cm and even to 1cm.

```
\setlength{\oddsidemargin}{0cm}
\setlength{\evensidemargin}{0cm}
```

Set width and height for the text. Width to 15cm and height to 24.5cm.

```
\setlength{\textwidth}{15cm}
\setlength{\textheight}{24.5cm}
```

Get half a centimeter back from the topmargin.

```
\setlength{\topmargin}{-1.5cm}
```

Don't use chapter numbers in sections, thus making them looking like sections in a classic article (1 instead of the default 0.1)

```
\def\thesection{\arabic{section}}
```

Allow table of contents to go up to sub-sections

```
\settocdepth{subsection}
```

And numbering up to subsubsections

```
\setsecnumdepth{subsubsection}
```

Set the head styles to komalike (the other nice style is memman).

```
\headstyles{komalike}
```

For lists, memoir provides different layouts. We use tightlists here, but can switch that to firmlists if needed

```
\tightlists
%\firmlists
```

What are these for? I forgot...

```
\midsloppy
\raggedbottom
```

Set parindent to 0pt and parskip to 0.2pt.

```
\parindent0pt
\setlength{\parskip}{0.2cm}
```

13.4.3 Misc Settings

Do an index.

```
\makeindex
```

Load the bibliographic information using the SKB.

```
\skbbibtex
```

Before we start with the actual document, we want the title slide and the table of contents on the first page.

```
\AtBeginDocument{
  \resizebox{\textwidth}{!}{\includeslide{title}}
  \bigskip
  \tableofcontents*
  \bigskip
  \newpage
}
```

There is no code for `\AtEndDocument`, so we are done now.

⟨/skblncsbeamer⟩

13.5 The Class skblncsppt

This class is an example on how to use the SKB with memoir. I use skblncsppt for handouts (anotated slides) based on Microsoft's PPT. Reason for that is that the PDF export and print routines in Microsoft Office 2010 no longer support vector images for the slide thumbnails, which renders handouts almost useless. So I do print the PPT slides into PDF (screen resolution, that way one avoids frames around the slides), and then LaTeX to generate handouts. Using this class as a template, one can easily write other classes or change/overwrite the settings done here.

First, we announce the package and the font definition file.

```
731 ⟨*skblncsppt⟩
732 \NeedsTeXFormat{LaTeX2e}
733 \ProvidesClass{skblncsppt}[2010/08/04 The SKB LNCS PPT class v0.5]
```

Now we load the memoir class with the following options:

- 9pt - for 9 point font size
- a4paper - I am European, so A4 paper makes sense here
- extrafontsizes - tbd
- twoside - I want my articles to be set with different even/odd pages
- onecolumn - I don't necessarily like 2-columns for my articles
- openright - tbd
- article - use memoir as if it is an article

```
734 \LoadClass[9pt,a4paper,extrafontsizes,twoside,onecolumn,openright,article]{memoir}
```

Load the SKB.

```
735 \RequirePackage{skb}
```

13.5.1 Loaded Packages

I prefer BibLaTeX over plain BibTeX, and other parts of the SKB (such as the LAMP server) produce BibLaTeX. The options are:

- style - is set to alphanumeric, much better to find/remember references. If writing for IEEE or LNCS, numeric would be the prefered option.
- sorting - is set to none, not needed here.
- hyperref - I want to have hyperef with my citations

```
736 \RequirePackage[style=alphabetic,sorting=none,hyperref]{biblatex}
```

Load the acronym package and print only the acronyms actually used in the document. This might move into the SKB package later

```
737 \RequirePackage[printonlyused]{acronym}
```

Load a view packages that I tend to use quite often:

- etoolbox - etoolbox
- comment - Add comments to your LaTeX files
- graphicx - Enhanced graphic support, with key/value interface for include graphics
- longtable - Helps with tables that span multiple pages
- colortbl - Allows coloured cells in tables

```
738 \RequirePackage{etoolbox,comment,graphicx,longtable,colortbl}
```

And some more packages needed quite often:

- textcomp - Special characters, such as ® and ©
- gensymb - Generic characters (math and text mode), such as °, ℃, ‰, µ and Ω
- wasysym - Adds characters from *wasy* font, such as ☺, ⊠ and ♌
- units - Typeset units correctly (and produce 'nice' fractions), such as $10\,{}^{\mathrm{m}}/_{\mathrm{s}}$ and $^1/_2$
- float - Improves interface for floating environments (such as figures, tables)
- xmpmulti - tbd

```
\RequirePackage{textcomp,gensymb,wasysym,units,xmpmulti,float}
```

The xcolor package provides driver independent access to all sorts of colour tins, shades, tones and mixes. I like x11names, as you can tell.

```
\RequirePackage[x11names]{xcolor}
```

The hyperref package provides layout for hyper references, such as URLs and references within a document, such as acronyms, citations and the table of contents. We use the option colorlings and then provide the colors we prefer for links (linkcolor), citations (citecolor) and URLs (urlcolor).

```
\RequirePackage[colorlinks,%
                linkcolor=AntiqueWhite4,%
                citecolor=SeaGreen4,%
                urlcolor=RoyalBlue3%
               ]{hyperref}
%\RequirePackage[colorlinks,linkcolor=blue]{hyperref}
```

13.5.2 Memoir Options

Not sure, but I don't think semi-iso-pages are good. So not used right now.

```
%\semiisopage
```

We do want to list files.

```
\listfiles
```

Change the margins for even and odd pages. Odd to 0cm and even to 1cm.

```
\setlength{\oddsidemargin}{0cm}
\setlength{\evensidemargin}{0cm}
```

Set width and height for the text. Width to 15cm and height to 24.5cm.

```
\setlength{\textwidth}{15cm}
\setlength{\textheight}{24.5cm}
```

Get half a centimeter back from the topmargin.

```
\setlength{\topmargin}{-1.5cm}
```

Don't use chapter numbers in sections, thus making them looking like sections in a classic article (1 instead of the default 0.1)

```
\def\thesection{\arabic{section}}
```

Allow table of contents to go up to sub-sections

```
\settocdepth{subsection}
```

And numbering up to subsubsections

```
\setsecnumdepth{subsubsection}
```

Set the head styles to komalike (the other nice style is memman).

```
\headstyles{komalike}
```

For lists, memoir provides different layouts. We use tightlists here, but can switch that to firmlists if needed

```
\tightlists
%\firmlists
```

What are these for? I forgot...

```
\midsloppy
\raggedbottom
```

We want ruled pages and arabic page numbering.

```
\pagestyle{ruled}
\pagenumbering{arabic}
```

13.5.3 Misc Settings

Do an index.

```
\makeindex
```

Load the bibliographic information using the SKB.

```
\skbbibtex
```

There is no code for `\AtBeginDocument` and `\AtEndDocument`, so we are done now.

⟨/skblncsppt⟩

14 History and Change Log

14.1 v0.10 from 06-Jul-2010

- first source forge release of the skb

- at this stage a collection of .sty and .tex files
- documentation in a separate pdf file
- included acronym list

14.2 v0.20 from 08-Jul-2010

- first LaTeX package version of the skb
- no changes in the documentation and no change in commands
- removed acronym list

14.3 v0.30 from 14-Jul-2010

- First dtx release of the skb, including the package and all classes introduced in v0.2
- Integrated parts of the v0.1 pdf as documentation and added documentation for many commands (not finished though)
- Re-write of all load commands (publish, repository, figures, acronyms, bib) and rename of all old load commands, new command names use only lowercases in their names
- In rewrite, many commands could be removed w/o losing their functionality
- New Commands:
  - `\skbfigure` – load figures with some options
  - `\skbinput` – load files with some options
  - `\skbheading` – set heading text in a file loaded
  - `\skbheadingudc` – set heading relatively to the last heading level (up, down, current) (experimental)
  - `\skbem` – emphasise code using options
  - `\skbacft` – rename of `\SkbAcFT`
  - `\skbacronyms` – rename of `\SkbLoadAcronyms`
  - `\skbbibtex` – rename of `\SkbLoadBibtex`
  - environment `skbnotelist` – itemize list with `\parskip 0` and `\itemskip 0`
  - environment `skbnoteenum` – enumerate list with `\parskip 0` and `\itemskip 0`
- Replaced Commands:
  - `\SkbSetTitle` $\mapsto$ replaced by `\skbheading`
  - `\SkbFigure` $\mapsto$ removed, closest is `\skbfigure` (but changed behaviour!)
  - `\listingInline` $\mapsto$ replaced by `\skbem[code]`
  - `\SkbEmIT` $\mapsto$ replaced by `\skbem[italic]`
  - `\SkbEmBF` $\mapsto$ replaced by `\skbem[bold]`
  - `\SkbAcFT` $\mapsto$ replaced by `\skbacft`
  - `\SkbLoadAcronyms` $\mapsto$ replaced by `\skbacronyms`
  - `\SkbLoadBibtex` $\mapsto$ replaced by `\skbbibtex`
  - `\SkbLoadRepository` $\mapsto$ replaced by `\skbinput[from=rep]`

  - `\SkbLoadPublish` ↦ replaced by `\skbinput[from=pub]`
  - `\SkbItemizeBegin` ↦ replaced by `\begin{skbnotelist`
  - `\SkbItemizeEnd` ↦ replaced by `\end{skbnotelist`
  - `\SkbEnumerateBegin` ↦ replaced by `\begin{skbnoteenum`
  - `\SkbEnumerateEnd` ↦ replaced by `\end{skbnoteenum`
  - `\SkbFigureBeamerTextWidth` ↦ replaced by `\skbfigure[width=##]`
  - `\SkbFigureBeamerTextHeight` ↦ replaced by `\skbfigure[height=##]`
  - `\SkbFigureBeamerNoResize` ↦ replaced by `\skbfigure[]`
  - `\SkbFigureBeamerTextWidthPDFMulti` ↦ replaced by `\skbfigure[multiinclude=##]`

14.4 v0.31 from 20-Jul-2010

- fixed space problem in `\skbem`
- added error handling to the options skbconfig and skbheading
- added error handling for skbinput related macros
- separated documentation, skb.dtx is now using itself to create the documentation
- removed old code: DeclareOptions (none declared)
- changed a lot in the documentation
- prepare for CTAN submission, i.e. adding README and other things
- New Commands:
  - `\skbconfig` – change the path/file options
  - `\skbsubject` – add subject information for PDF
  - `\subkeywords` – add keyword information for PDF
  - `\skbpdfinfo` – generate PDF information
- Changed Commands:
  - `\skbfigure` – added option for position, moved caption/label from argument to option
  - `\title` – re-newed to store PDF info information (experimental)
  - `\author` – re-newed to store PDF info information (experimental)
- Replaced Commands:
  - `\SkbCodeInline` ↦ replaced by `\skbcode`

14.5 v0.32 from 20-Jul-2010

- fastest re-release, I had built-in some problems and excluded important code in v0.31, fixed now

14.6 v0.4 from 21-Jul-2010

- major re-write of the kernel subsequently the documentation. Most internal macros will have been changed or removed, some are added. Also re-arranged the macros in the dtx file to (hopefully) optimise the documentation

- added input for skb.cfg and skblocal.cfg to overwrite package options with configuration files
- added skb.cfg to the distribution
- New Commands:
  - `\skbpathroot` – returns current root path
  - `\skbfileroot` – returns root/#1
  - `\skbfileacr` – returns current acronym path and file
  - `\skbfilebib` – returns current bibtex path and file
  - `\skbpathbib` – returns current bibtex path
  - `\skbfilerep` – returns rep/#1
  - `\skbfilepub` – returns pub/#1
  - `\skbfilefig` – returns fig/#1
  - `\skbfilesli` – returns sli/#1
  - `\skboptionsused` – prints a warning with change log of otptions and current values
- Changed Commands:
  - `\skbconfig` – added parameter to identify origin of the configuration change
- Replaced Commands:
  - `\SkbPathBib` ↦ replaced by `\skbpathbib`
  - `\SkbPathFig` ↦ replaced by `\skbfilefig`

14.7 v0.5 from 04-Aug-2010

- added example describing how the SKB uses itself to create parts of its documentation
- removed the redefinition of `\title` and `\author`, since they intererred with the beamer package definitions of these macros. added `\skbtitle` and `\skbauthor` instead.
- added RequiredPackage in the class skbbeamer before loading beamer to load xcolors with x11names
- added test for nemoir class: if loaded, then skbnotelist and skbnoteenum will have no effect; if not loaded, then the package booktabs will be loaded (for top/mid/bottomrule
- added test for beamer package: depending if beamer or beamerarticle are loaded, the SKB will initialise a few newe ifs
- added required package dirtree, and redefinition of some dirtree styles
- added two options to the SKBpackage: beameranim and beamernoanim
- added the package versions with the environments: skbmodetext, skbmodenote and skbmodeslide
- added the package optional with the options: text, note, slide, anim and noanim
- internally, the package optional also provides memoir

- changed the documentation, moved manual description to user guide in folder /doc, moved history.tex into the dtx file and changed most of the actual documentation (still not finished though)
- skbbeamer: corrected load of beamer package
- skblncsbeamer: moved load of skb after beamerarticle to allow skb to create proper options
- added `\providecommand` for `\escribeMacro` and `\cmd`, so that we can use the user-guide in the dtx and stand-alone
- added conditional load of skb.dtx in the driver
- changed the sequence of definitions in the dtx file, again, hopefully the last time
- Bug Fixes (SF=sourceforge):
  - SF#3032749 (skboptionsused doesn't work) – fixed, changed `\skb@setCfgVars`
  - SF#3032752 (history section for v0.4 has wrong date) – fixed, changed the heading
  - SF#3032754 (skb.cfg missing/empty) – fixed, changed the installer (skb.ins) to generate it and my local scripts to put it into /run
  - SF#3033124 (renewcommand title/author doesn't work) – fixed, no renewcommand anymore, two new commands to set author/title for pdfinfo
  - SF#3038935 (skbinput not working w/o from) – fixed, can load from root directory now
- New Commands:
  - `\skbtitle` – title for PDF info
  - `\skbauthor` – author for PDF info
  - `\skbslide` – load slides and annotations
  - `\skbslidecite` – for citations on slide annotation pages
- Changed Commands:
  - `\skbinput` – added option to load tex files from figures directory (option fig)
- Replaced Commands:
  - `\SkbLoadSlideNotes` ↦ replaced by `\skbslide` with option annotate and first argument only
  - `\SkbLoadSlideNotesDifferent` ↦ replaced by `\skbslide` with option annotate and both arguments
  - `\SkbLoadSlideNotesExtern` ↦ replaced by `\skbslide` with option annotate and both arguments and option notefrom set
  - `\SkbLoadSlideNotes` ↦ replaced by `\skbslide` without annotate and first argument only
  - `\SkbLoadSlideOnlyNotes` ↦ replaced by `\skbslide` with option annotate and second argument only
  - `\SkbSlideSource` ↦ replaced by `\skbslidecite`
  - `\SkbBeamerAnimtrue` ↦ replaced by options beameranim and beamernoanim for skbbeamer

– `\SkbBeamerAnimtrue` $\mapsto$ usage of this if replaced by `\opt` with anim and noanim