labbook.cls, a LaTeX class to typeset laboratory journals

Frank Küster

2003/05/20

Abstract

This class is designed to typeset laboratory journals with LaTeX that contain chronologically ordered records about experiments. Each day, the new pages are simply added at the end, and from the sectioning commands, an experiment index is generated to make it easier to find a particular experiment. Because `\frontmatter` has roman page numbering, it is easy to separately extend index, toc and list of abbreviations. The class is based on the KOMA-Script class `scrbook.cls`. Therefore all features of this class can be used – reading of `scrguide` (german) or `scrguien` (english), the documentation of KOMA-Script, is highly recommended.

Contents

Part I
User documentation

1 General information

1.1 Legal considerations

Writing of laboratory journals is sometimes regulated by law, often funding institutions have additional requirements. Thus it is usually *not* allowed to keep a lab journal in electronic form only; sometimes even hardcover books have to be used. On the other hand it seems to be common practice to keep notes on loose leafes, computer files or a jotter book and compile them in the hardcover lab journal in a more or less regular manner.

Therefore it seems tenable to work with an electronic version of the journal and a printout in parallel. If required, the printout can be clipped or bound, and a version on a CD-ROM should have equal value as evidence as a book. On the other hand, one gains a lot, especially with a PDF version: full text search, easy inclusion of graphs and pictures, possibility of hyperlinking to data files etc.

1.2 Features

basis class `labbook.cls` is based on the KOMA-Script-class `scrbook.cls`. Therefore it offers all the features of this class: changing the page layout, head- and footlines, paragraph markup, and changing appearance of floats and margin notes. It is highly recommended to read the documentation, scrguien.

Sectioning The lab journal is structured in a chronological way. Instead of `\chapter`, `\labday` is used. Within a day the text is sectioned according to experiments – this can be a complete measurement from design to evaluation, but also one test in a series that takes several days. `\experiment` is on the level of section, and there is `\subexperiment` which corresponds to `\subsection`. Below that, the usual sectioning commands can be used (although their usage doesn't seem very sensible to me, mind KOMA-script's `\minisec` command).

Index and table of contents Because of the chronological structure, a traditional table of contents is not sufficient, especially if investigations in a series of experiments are conducted on different days. Therefore, an index is created. It is built from the toc entries of the sectioning commands. One can also define abbreviations to use in the argument of `\experiment` and `\subexperiment`. Thus, the consistent creation of the index is much easier. Furthermore, you can create multiple index entries for one experiment.

hyperref integration The class works together well with `hyperref.sty` to produce PDF files with navigation features.

2 Usage

2.1 Initialization

If you want to use hyperref.sty, you *must* specify the class option `hyperref` like this

```
\documentclass[hyperref]{labbook}
```

after that you can load hyperref and other packages in the order appropriate for your document. If you load hyperref.sty without giving the option, hyperref will overwrite some changes this class makes to LaTeX internals. If you use hyperref, you should also consider the KOMA option `idxtotoc`: The index will not only be listed in the table of contents, but also in the PDF bookmarks.

Besides that, all options you specifiy are just handed on to `scrbook.cls`. You might want to use `openany` to allow new labdays to begin on any page.

3 Sections commands

3.1 \labday

`\labday` `labday` can be used to create an unnumbered heading formatted like a chapter. Its text is put into the table of contents and used as running title – usually one would just use the date, perhaps with the weekday. It should be used only within mainmatter and will take care of index entries for the lower sectioning levels (see below). It calls `\addchap` internally, there is no starred form or optional argument to `\labday`.

3.2 \experiment and \subexperiment: The simple usage

`\experiment` Within a `\labday`, numbered headlines for single experiments can be created with

```
\experiment[<short form>]{long form}
```

Its text (or that of the optional argument) is not only typeset in the table of contents and page head, but also in the index. This eases the orientation, particularly if experiments are done several times or take longer than one day. The index entries specify the page range of the whole experiment, not only the starting page; and if the same experiment is repeated on different days that fall on subsequent pages, one common index range is generated.

Note that you cannot use commas[1] in the optional argument (see below). If you have to, enclose the optional argument in additional curly braces:

[1] In fact, the string `@--@` is not allowed, too – see the implementation notes below.

```
\experiment[{one, two, three}]{The one, the two and the three}
```

\subexperiment Below \experiment you can use \subexperiment which works in the same way and produces sub-indexentries. It is intended for sections like "design, realization, evaluation" or "preparation, purification, measurement".

3.3 Advanced Usage: Experiment Abbreviations

\newexperiment If one uses slightly different wording (or spelling) in two occurences of the same experiment, they will get different index entries. To avoid this, one can define abbreviations for frequently used experiment headlines, toc and index entries. This is done with the macro

`\newexperiment{<abbrev>}[<short form>]{<long form>}`

Here, <abbrev> is the abbreviation that can be given later to make LaTeX use the <long form> and <short form>. The short form is for index, table of contents and running title, and giving it is optional. When using the abbreviation, specify it *without* prepending a backslash, i.e. \experiment{abbrev}. Abbreviations may contain any char except the backslash, the tilde (~), commas and spaces.

\newsubexperiment For \subexperiment, there is an analogous macro, \newsubexperiment.

If you try to define an abbreviation that has already been used, you will get an error message. You can use the same abbreviation for one experiment and one subexperiment entry (although doing this may cause confusion, not to TeX, but to you). If you leave out the optional argument, the long form is also used for index and table of contents.

Usually you just type \experiment{abbrev}, but you can also combine a varied text in the long form with an abbreviation in the optional argument: \experiment[abbrev]{varied text}, to make sure the index stays simple. You cannot use a free text optional argument with an abbreviation in the mandatory argument (because the abbreviation yet has an associated index entry). But it is, in principle, possible to use two different abbreviations in optional and mandatory argument - but only as long as both yield the same index entry.

3.4 Fancy stuff: Multiple index entries for experiments

Sometimes one performs corresponding working steps of different experiments in parallel – this complicates index entries. Consider that by some screening method you have identified the substances A152 and B96 from a combinatorial library as promising drugs against some disease. The next things to do is to verify their exact structure, composition or sequence and establish a medium-scale preparation protocol for further characterization. Probably you can save time by doing some of these steps in parallel, but you will end up with index entries like "A152 and B96, sequencing" – and two months later you will have to remember wether you sequenced B96 together with A152 or rather, the following

week, with A43 and C12. Therefore it would be nice to get two index entries for the experiment "Sequencing of A152 and B96", namely "A152, sequencing" and "B96, sequencing". And you can have exactly that.

\experiment
\subexperiment

In fact the syntax for both `\experiment` and `\subexperiment` allows for a comma separated list in the optional argument. The first element will be used for table of contents and page head, and the following elements will produce index entries. Suppose you have defined the abbreviations A152-seq and B96-seq, you can thus say:

```
\experiment[Sequencing of A152 and B96, A152-seq, B96-seq]{Sequencing
  of inhibitor candidates A152 and B96}
```

and get what you want. Spaces before and after the commas will be ignored. You're free to use abbreviations or free text anywhere, but for the index entries only abbreviations really make sense.

4 Example

For further explanation, please refer to the example file examplen.tex that is generated by

```
latex labbook.ins
```

Part II
Implementation

1 ⟨*labbook⟩

5 Initialization

5.1 Options

Some packages the user might use have to be loaded before hyperref, so this class cannot load it internally. However, we have to take care of it because of two reasons: One is that we need to provide it with some macros for our new sectioning levels, the more important is that we cannot redefine `\addcontentsline` before hyperref has done it. But we can't simply delay this until after hyperref has been loaded, since this might never occur.

Therefore the user has to specifiy wether she intends to use it, using the option `hyperref`.

```
2 \newif\ifwe@use@hyperref\we@use@hyperreffalse
3 \DeclareOption{hyperref}{\we@use@hyperreftrue}
```

The rest is just passed to `scrbook.cls`:

```
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{scrbook}}
\ProcessOptions\relax
\LoadClass{scrbook}
```

5.2 Load the makeidx package

```
\RequirePackage{makeidx} \makeindex
```

6 Experiment abbreviations

To facilitate the creation of the index (specifically to avoid that little differences or typos mess it up), the user may define abbreviations for his experiment (and subexperiment) headings. This can be done using the macro `\newexperiment`. It's first argument is the abbreviation, then comes an optional short form for index entries, table of contents and running titles. Last comes the mandatory title argument.

Then we check wether `\newexperiment` has been called with an optional argument, and call the respective commands. `\@ifnextchar` looks for the first character after the first argument, and keeps it. Thus, the following macros get the same arguments that `\newexperiment` had.

```
\def\newexperiment#1{%
  \@ifnextchar [{\opt@arg@newexperiment{#1}}%
    {\nopt@arg@newexperiment{#1}}%]
}
```

The long and short forms for *abbrev* are assigned to `\long@<abbrev>` and `\short@<abbrev>`, respectively. After checking wether *abbrev* has not yet been used, the second argument is assigned to the long and short form for the case when there was no optional argument. In case of an optional argument, this is used for the short form.

```
\def\nopt@arg@newexperiment#1#2{%
  \@ifundefined{long@exp@#1}{%
    \@namedef{long@exp@#1}{#2}%
    \@namedef{short@exp@#1}{#2}%
    }{%
    \ClassError
      {labbook}
      {experiment abbreviation yet defined}
      {The abbreviation for an experiment that you wanted to define
      with this command has already been defined.}%
    }
}
\def\opt@arg@newexperiment#1[#2]#3{%
  \@ifundefined{long@exp@#1}{%
    \@namedef{long@exp@#1}{#3}%
    \@namedef{short@exp@#1}{#2}%
  }{%
```

```
    \ClassError
      {labbook}
      {experiment abbreviation yet defined}
      {The abbreviation for an experiment that you wanted to define
      with this command has already been defined.}%
  }
}
```

The same is done for `\subexperiment`. Any subexperiment abbreviation may be used within any experiment.

```
\def\newsubexperiment#1{%
  \@ifnextchar [{\opt@arg@newsubexperiment{#1}}%]
    {\nopt@arg@newsubexperiment{#1}}%
}%
\def\nopt@arg@newsubexperiment#1#2{%
  \@ifundefined{long@subexp@#1}{%
    \@namedef{long@subexp@#1}{#2}%
    \@namedef{short@subexp@#1}{#2}%
    }{%
    \ClassError
      {labbook}
      {experiment abbreviation yet defined}
      {The abbreviation for an experiment that you wanted to define
      with this command has already been defined.}%
    }%
}%
\def\opt@arg@newsubexperiment#1[#2]#3{%
  \@ifundefined{long@subexp@#1}{%
    \@namedef{long@subexp@#1}{#3}%
    \@namedef{short@subexp@#1}{#2}%
  }{%
    \ClassError
      {labbook}
      {experiment abbreviation yet defined}
      {The abbreviation for an experiment that you wanted to define
      with this command has already been defined.}%
  }%
}%
```

Note that the usage is `\experiment{`*`abbrev`*`}`, *not* `\experiment{\`*`abbrev`*`}`

7 Defining internal macros

We first define some internal helper macros that we will use for different purposes.

7.1 Macros for argument parsing

7.1.1 Parsing a comma separated list

The new sectioning commands can be used with a comma separated list of items in the optional argument (see below, **??**), and we will be constructing a similar list if there is one more index entry for one sectioning command. To parse this list, we use code that is essentially taken from `keyval.sty` (page 4 in its manual).

First we need some helper macros. `\FK@@sp@def` defines the control sequence in its first argument to expand to its second argument, but with any leading or trailing whitespace removed.

```
\def\@tmpA#1{%
\def\FK@@sp@def##1##2{%
  \futurelet\FK@tempa\FK@@sp@d##2\@nil\@nil#1\@nil\relax##1}%
\def\FK@@sp@d{%
  \ifx\FK@tempa\@sptoken
    \expandafter\FK@@sp@b
  \else
    \expandafter\FK@@sp@b\expandafter#1%
  \fi}%
\def\FK@@sp@b#1##1 \@nil{\FK@@sp@c##1}%
  }%
\def\FK@@sp@c#1\@nil#2\relax#3{\FK@toks@{#1}\edef#3{\the\FK@toks@}}%
\newtoks\FK@toks@%
\@tmpA{ }%
```

Now we define a way to parse the optional argument and break it at every comma. `\fk@getcommasep@list` is a wrapper macro. It executes its first argument once before the list is read, assigns its further arguments to actions that will be done while going through the list, and then calls `\fk@@getcommasep@list` which takes as its argument the first element of the list after it, i.e. everything up to the first comma. It then works through the previously defined actions with this element and then calls itself recursively, taking the next element as its argument. `\fk@getcommasep@list` has to be called with `<list>,\relax,`. Therefore, when all elements of the list have been used, `\relax` is the next, and the recursion ends.

```
\newcounter{fk@commasep@argnumber}%
\def\fk@getcommasep@list#1#2#3#4#5#6{%
  \setcounter{fk@commasep@argnumber}{0}%
  #1
  \def\fk@commasep@beforebranch{#2}%
  \def\fk@commasep@firstelement{#3}%
  \def\fk@commasep@furtherelements{#4}%
  \def\fk@commasep@afterlastelement{#5\empty}%
  \def\fk@commasep@aftereachelement{#6}%
  \fk@@getcommasep@list
}
\def\fk@@getcommasep@list#1,{%
```

```
  \stepcounter{fk@commasep@argnumber}%
  \fk@commasep@beforebranch%
  \ifx\relax#1%
    \fk@commasep@afterlastelement%
```

The counter has been incremented, and if we are already after the last element (so the argument was `\relax`), then `\fk@commasep@afterlastelement` (the fifth argument to the wrapper macro) is executed.

The `\else`-branch means we have a real list element. We assign it (with whitespace removed) to `\fk@commasep@arg`. Then we check the counter wether we are working with the first element of the list, or with subsequent elements, and execute the respective commands (arguments number 3 and 4 to the wrapper macro).

```
  \else%
    \FK@@sp@def\fk@commasep@arg{#1}%
    \ifnum\c@fk@commasep@argnumber=1%
      \fk@commasep@firstelement%
    \else%
      \fk@commasep@furtherelements%
    \fi%
```

After that, the sixth argument is executed which has been assigned to `\fk@commasep@aftereachelement`, and the macro calls itself again to take the next element.

```
    \fk@commasep@aftereachelement%
    \expandafter\fk@@getcommasep@list%
  \fi%
}
```

Index entries for a sectioning level may contain commas, therefore using a comma separated list for them is not a good idea. We just copy the above definitions, but use `@--@` as the delimiter. Should you ever have to use this in an index entry, you need to define a command that typesets it and put this command in your optional argument, or the abbreviation definition.

```
\newcounter{fk@atdashsep@argnumber}%
\def\fk@getatdashsep@list#1#2#3#4#5#6{%
  \setcounter{fk@atdashsep@argnumber}{0}%
  #1
  \def\fk@atdashsep@beforebranch{#2}%
  \def\fk@atdashsep@firstelement{#3}%
  \def\fk@atdashsep@furtherelements{#4}%
  \def\fk@atdashsep@afterlastelement{#5\empty}%
  \def\fk@atdashsep@aftereachelement{#6}%
  \fk@@getatdashsep@list
}
\def\fk@@getatdashsep@list#1{%
  \stepcounter{fk@atdashsep@argnumber}%
  \fk@atdashsep@beforebranch%
  \ifx\relax#1%
```

```
    \fk@atdashsep@afterlastelement%
  \else%
    \FK@@sp@def\fk@atdashsep@arg{#1}%
    \ifnum\c@fk@atdashsep@argnumber=1%
      \fk@atdashsep@firstelement%
    \else%
      \fk@atdashsep@furtherelements%
    \fi%
    \fk@atdashsep@aftereachelement%
    \expandafter\fk@@getatdashsep@list%
  \fi%
}
```

7.1.2 Parsing lists in the optional argument

Now we define a macro that uses `\fk@getcommasep@list` to parse the comma separated list in the optional argument of `\experiment` and `\subexperiment` (see **??**). Its only declared argument is the sectioning level specifier (`exp` or `subexp`), but it should be called with the comma separated list (ending with `\relax,`) following the specifier. This specifier is stored in the macro `\fk@explevel` to make its usage more clear.

First `\ifmore@thanone@item` is defined. It will be used to conditionally trigger actions after the last element. Before starting to iterate over the list, i.e. in the first argument to `\fk@getcommasep@list`, we set this to false.

```
\newif\ifmore@thanone@item%
\def\fk@parse@optarg{%
  \fk@getcommasep@list{%
    \more@thanone@itemfalse%
  }{%
  }{%
```

There's nothing to be done on every iteration before branching, so the second argument to `\fk@getcommasep@list` is empty. The third is executed for the first element in the list. We simply define the macro `\fk@current@tocentry` as this first element for later use. `\@onelevel@sanitize` changes `\fk@commasep@arg` to consist only of strings, not command sequences. This is necessary because hyperref will give an error with certain command sequences in bookmarks etc, e.g. with `\textit` which expands to `\protect\textit`. Wether this item will only be used for the toc and running title, or wether it additionally is put into the index (namely if there is only this first argument) cannot be decided now.

```
    \@onelevel@sanitize{\fk@commasep@arg}%
    \protected@edef\fk@current@tocentry{\fk@commasep@arg}%
  }{%
```

If there is at least a second element[2], we set `\ifmore@thanone@item` to

[2]note that it doesn't make sense to use a list with two elements - there will be only one index entry, so you can just as well use only one. The only difference will be that you get the possibility to have different wording in the index entry *and* the toc entry and the section heading, which I would try to avoid.

`true`, then assign the currently processed list element to `\fk@currentarg`, appending a space only after we have made sure there will be no problems with fragile commands. Then we check wether it is an abbreviation for the current sectioning level. According to the result, we create the text to put in the index (`\fk@current@arg` itself or the text the abbreviation stands for). Then we call `\fk@buildindexlist` (see below) with it to add it to the index list for the current sectioning commands.

```
    \more@thanone@itemtrue%
    \protected@edef\fk@currentarg{\fk@commasep@arg}%
    \@onelevel@sanitize{\fk@currentarg}%
    \protected@edef\fk@currentarg@withspace{\fk@currentarg\space}%
    \expandafter\fk@checkifabbrev@arg%
      \fk@currentarg@withspace&{long@\fk@explevel}%
    \ifabbrev@defined%
      \expandafter\protected@edef%
          \csname fk@current@\fk@explevel name\expandafter%
          \endcsname{%
            \csname short@\fk@explevel @\fk@currentarg\endcsname}%
      \fk@buildindexlist{%
        \csname short@\fk@explevel @\fk@currentarg\endcsname}%
    \else%
      \expandafter\protected@edef%
          \csname fk@current@\fk@explevel name\endcsname{%
            \fk@currentarg}%
      \expandafter\fk@buildindexlist{\fk@currentarg}
    \fi%
  }{%
```

After the last element, the procedure of sanitizing, space appending and abbreviation checking is repeated. Because this is done after the last element, the result of the check will not be overwritten by other index elements. If there was only one element in the optional argument, we have to add the content of `\fk@current@tocentry` to the (still empty) indexlist.

In this case, there is nothing to be done after each processed list element, so the last argument to `\fk@getcommaseplist` is empty.

```
    \protected@edef\fk@currentarg{\fk@current@tocentry}%
    \@onelevel@sanitize{\fk@currentarg}%
    \protected@edef\fk@currentarg@withspace{\fk@currentarg\space}%
    \expandafter\fk@checkifabbrev@arg%
      \fk@currentarg@withspace&{long@\fk@explevel}%
    \ifmore@thanone@item\else%
      \ifabbrev@defined%
        \fk@buildindexlist{%
          \csname short@\fk@explevel @\fk@currentarg\endcsname}%
      \else%
        \expandafter\fk@buildindexlist{\fk@currentarg}%
      \fi%
    \fi%
```

```
  }{%
  }%
}
```

7.1.3 Building and using the list of index entries

The text to be indexed for every sectioning command is kept in a comma separated list, even if there is only one element. This list is build by prepending the new element to the expansion of the list. First, the lists (for experiments and subexperiments) are defined (as empty lists). The macros that call `\fk@buildindexlist` have to properly set `\fk@explevel` to `exp` or `subexp`, so that the new element gets into the right list.

```
\def\fk@exp@indexlist{}%
\def\fk@subexp@indexlist{}%
\def\fk@buildindexlist#1{%
  \def\@tmpA{exp}
  \ifx\fk@explevel\@tmpA
    \protected@edef\fk@exp@indexlist{#1\fk@exp@indexlist}
  \else
    \protected@edef\fk@subexp@indexlist{#1\fk@subexp@indexlist}
  \fi
}
```

To write index entries for each list element, we use `\fk@getcommasep@list` again. Here there is no difference between first and further elements, so we just define commands in the sixth argument, which is processed once for every element. The list never contains abbreviations, but always the text itself, so we just call `\fk@@writeindex` with the element.

```
\def\fk@useindexlist{%
  \fk@getatdashsep@list{}{}{}{}{}{%
    \fk@@writeindex{\fk@atdashsep@arg}%
  }%
}%
```

`\fk@@writeindex` handles the differences in index writing for experiments and subexperiments. If a new experiment is started (or closed), then an index for every list element has to be written:

```
\def\fk@@writeindex#1{%
  \def\@tmpA{exp}%
  \ifx\fk@explevel\@tmpA%
    \fk@writeindex{#1}%
  \else%
```

If we are writing index entries for a subexperiment, things get more complicated, because the experiment *and* the subexperiment might have more than one associated index entry. Therefore we have to iterate over both `\fk@exp@indexlist` and `\fk@subexp@indexlist`. To achieve this, for a subexperiment (where `\fk@explevel` is `subexp`), `\fk@useindexlist` will be

called with \fk@*exp*@indexlist, and \fk@parselevel is set to exp. When \fk@openindex is called with explevel subexp, but parselevel exp, its argument is assigned to \fk@current@expname, the parselevel is changed to subexp, and \fk@useindexlist is called again with \fk@*subexp*@indexlist. When doing this, \fk@openindex will be called again, but since now the parselevel is subexp, we get into the other branch. After processing \fk@subexp@indexlist is finished, we have to continue processing the next elements in \fk@exp@indexlist. Therefore we reset the parselevel to exp.

```
    \ifx\fk@parselevel\@tmpA% we are iterating over the current list
                            % of experiments
      \protected@edef\fk@current@expname{#1}
      \def\fk@parselevel{subexp}
      \expandafter\fk@useindexlist\fk@subexp@indexlist\relax%
      \def\fk@parselevel{exp}
    \else
```

When processing the subexperiment's index list, we get into this branch, and now for every subexperiment index element an entry is made for the current meaning of \k@current@expname.

```
      \fk@writeindex{\fk@current@expname!#1}%
    \fi
  \fi%
}%
```

The previous procedure made use of \fk@writeindex which is never defined itself. It is \let to \fk@openindex or \fk@closeindex, depending on where it is called. The first writes index entries which open a range, i.e. \index{text|(}, the second closes the ranges again (\index{text|)}).

```
\def\fk@openindex#1{\index{#1|(}%)
\def\fk@closeindex#1{\index{#1|)}}%)
```

7.2 Checking for abbreviation usage

The next few macros are used to check wether an abbreviation or a text was used as the argument of \(sub)experiment. First a little helper macro:

```
\def\muST@bE@emPTy{\message{Numquam videbor}}%
```

\fk@checkifabbrev@arg is defined so that its first argument ends with a space and the second only at the &-sign; it is called with a space appended to the item to check. Thus, if the second argument isn't empty, there was a space in the item itself, and it cannot be an abbreviation. If the second argument is empty, we check wether we have an abbreviation using \fk@checkfirst:

```
\newif\ifabbrev@defined%
\def\fk@checkifabbrev@arg #1 #2&#3{%
  \ifx\muST@bE@emPTy#2\muST@bE@emPTy%
    \protected@edef\@tmpA{#1\space}%
    \expandafter\fk@checkfirst\@tmpA{#3}%
```

```
  \else%
    \abbrev@definedfalse%
  \fi%
}%
```

The first argument of `\fk@checkfirst` ends at the first space - note that `\fk@currentarg` was defined with a space appended. This is necessary because we have to be able to cut `\fk@currentarg` in words, not in single characters. The second argument to `\fk@checkfirst` specifies the level we are on (experiment or subexperiment).

The macro now checks wether the requested abbreviation has been defined, and sets the conditional accordingly:

```
\def\fk@checkfirst #1 #2{%
  \@ifundefined{#2@#1}{\abbrev@definedfalse}{\abbrev@definedtrue}%
}%
```

7.3 Closing the index entries

This macro, `\fk@close@labindex`, will be used in a couple of circumstances to close open index ranges. First we make `\fk@writeindex` produce entries that close the index ranges. If there has been no experiment on the current `\labday`, then `\fk@explevel` is undefined, and we don't do anything. Otherwise the experiment level is determined, and `\fk@@close@labindex` is called appropriately.

```
\def\fk@close@labindex{%
  \let\fk@writeindex\fk@closeindex%
  \@ifundefined{fk@explevel}{}{%
```

If `\fk@explevel` is `exp`, then we need to close any open experiment *and* subexperiment index ranges. We first handle the subexperiment. Before we call the macro that does it, we set `\fk@explevel` to `subexp` and `\fk@parselevel` to `exp`, as required by `\fk@@writeindex` (see **??**), and after that we reset it to `exp` to close the experiment ranges themselves. If `\fk@explevel` is `subexp`, however, we just call the closing macro:

```
    \def\@tmpA{exp}%
    \ifx\fk@explevel\@tmpA%
      \def\fk@parselevel{exp}%
      \def\fk@explevel{subexp}%
      \fk@@close@labindex%
      \def\fk@explevel{exp}%
      \fk@@close@labindex%
    \else%
      \fk@@close@labindex%
    \fi%
  }%
}%
```

`\fk@@close@labindex` calls `\fk@useindexlist` – always with the argument `\fk@exp@indexlist` – and then flushes the list for the current explevel.

```
\def\fk@@close@labindex{%
  \expandafter\fk@useindexlist\fk@exp@indexlist\relax%
  \expandafter\def\csname fk@\fk@explevel @indexlist\endcsname{}%
}
```

8 Defining the new sectioning commands `\labday` and `\(sub)experiment`

First we define the counters. Footnote numbers are reset every day.

```
\newcounter{labday}
\newcounter{experiment}[labday]
\newcounter{subexperiment}[experiment]
\@addtoreset{footnote}{labday}
```

If somebody uses `\subsubsection`, this has to be reset properly. The marks have to be let to `\@gobble` initially:

```
\@addtoreset{subsubsection}{subexperiment}
\let\experimentmark\@gobble
\let\subexperimentmark\@gobble
```

The level of sections is designed to be used to classify an experiment. The level below that, usually subsection, is redefined as `\subexperiment`. It should be used for things like "rationale", "preparations", or "evaluation", or the like, and will also get into the toc and index:

```
\setcounter{tocdepth}{3}
```

Thus, chapters, sections and subsections go into the toc besides labdays, experiments and subexperiments, but this should only be used for things in `\frontmatter` or at the start of `\mainmatter`.

Besides that, nothing is done in this class to format table of contents and index. This can be done individually with different packages on CTAN.

8.1 Defining experiment

```
\def\experiment{%
```

The first thing we do is close the index entry for the preceding `\experiment` and `\subexperiment`. Thus, this will be done before a potential page break. The macro `\fk@close@labindex` will be defined later, see **??**. Then we set `\iflower@sectionlevel` to `false` – this might have been set true by a previously used `\subsubsection`, were no indexing should be done.

```
  \def\fk@explevel{exp}
  \fk@close@labindex%
  \lower@sectionlevelfalse%
```

One can use the unstarred form with abbreviations, and with free text, too. However, if there is more than just text in the free text, e.g. a command like `\textit{...}`, then there will be an error message from TEX which is hard to understand. Therefore it is strongly recommended to use free text only with the starred form.

We descriminate between the starred and unstarred forms and first define the unstarred version: It checks wether there is an optional argument in square brackets and calls the respective macros:

```
\@ifstar{\@sexperiment}{\@experiment}%
}
\def\@experiment{%
  \@ifnextchar [{\opt@arg@experiment}{\nopt@arg@experiment}%
}
```

8.1.1 `\experiment` without optional argument

First the case of no optional argument is defined. Either the one argument is an abbreviation as defined by the user command `\newexperiment`, or it is of free form. Checking which is not straightforward, because we cannot simply put a whole sentence, possibly with markup macros, into an `\@ifundefined`-command.

```
\def\nopt@arg@experiment#1{%
  \def\fk@currentarg{#1 }\@onelevel@sanitize{\fk@currentarg}%
  \expandafter\fk@checkifabbrev@arg\fk@currentarg&{long@exp}%
```

The first line in the definition has the effect that `\fk@currentarg` contains the argument text, but everything is read as text, not as a macro, even if it looks like it. Then `\fk@checkifabbrev@arg` is called with this string of characters and (possibly) spaces as an argument and an arbitrary delimiter, here &. This macro, defined below, changes the conditional `\ifabbrev@defined`.

```
  \ifabbrev@defined%
```

The case with an abbreviation, so we have to manually assign short and long forms for `\@startsection`, and make sure that the short form gets into the index. Indexing is done by calling `\fk@buildindexlist` with the meaning of the abbreviation. It will automatically used by `\addcontentsline`. The actual arguments for `\@startsection` are just a copy of the `\section`-definition from scrbook.cls.

```
    \fk@buildindexlist{\csname short@exp@#1\endcsname}%
    \@startsection{experiment}{1}{\z@}%
         {-3.5ex \@plus -1ex \@minus -.2ex}%
         {2.3ex \@plus.2ex}%
         {\raggedsection\normalfont\sectfont\nobreak\size@section\nobreak}%
         [\@nameuse{short@exp@#1}]{\@nameuse{long@exp@#1}}%
  \else%
```

`\else` is the case where the user just uses the free form (or mistyped the abbreviation - but that will show up in the dvi/pdf file). `\@startsection` will `\@dblarg` the parameter itself.

```
    \expandafter\fk@buildindexlist{\fk@currentarg}
    \@startsection{experiment}{1}{\z@}%
         {-3.5ex \@plus -1ex \@minus -.2ex}%
         {2.3ex \@plus.2ex}%
         {\raggedsection\normalfont\sectfont\nobreak\size@section\nobreak}%
         {#1}%
  \fi%
}%
```

8.1.2 `\experiment` with an optional argument

Now comes the case where `\experiment` was called with an optional argument. The optional argument may consist of

- One arbitrary sentence with formatting *without* commas, or
- one previously defined abbreviation, or
- a comma separated list of items. The items should usually be abbreviations, the first may be an arbitrary sentence (with formatting).

In the latter case, the first one is used for the table of contents and the running title, but *not* for the index; the following are put in the index. All this is achieved by calling `\fk@parse@optarg`, after setting its parselevel to exp.

```
\def\opt@arg@experiment[#1]#2{%
  \fk@parse@optarg#1,\relax,%
```

The checks regarding abbreviations get more complicated here, we have to check the first item in the optional argument – this is done by the macro `\fk@parse@optarg` –, and the long argument.

```
  \ifabbrev@defined%
    \def\fk@currentarg{#2 }\@onelevel@sanitize{\fk@currentarg}%
    \expandafter\fk@checkifabbrev@arg\fk@currentarg&{long@exp}%
    \ifabbrev@defined%
```

This is the strange, but working case with two predefined forms. We have to check wether they are equal. In fact the user may use different abbreviations as long as the short forms expand to the same index/toc entry.

The check works like this: The first `\expandafter` delays the `\ifx` conditional until the next token, `\csname`, has been expanded. Expansion of `\csname`, however, scans for an `\endcsname`, but before it gets there (to the first one), it encounters the second `\expandafter`. Therefore, first the last `\csname` is expanded, yielding the macro `\short@exp@<second-abbrev>`, then the first is expanded, and we get
`\ifx\short@exp@<first-abbrev>\short@exp@<second-abbrev>`.

```
          \expandafter\ifx%
               \csname short@exp@\fk@current@tocentry\expandafter%
               \endcsname\csname short@exp@#2\endcsname%
            \@startsection{experiment}{1}{\z@}%
```

```
            {-3.5ex \@plus -1ex \@minus -.2ex}%
            {2.3ex \@plus.2ex}%
            {\raggedsection\normalfont\sectfont\nobreak\size@section\nobreak}%
            [\@nameuse{short@exp@\fk@current@tocentry}]%
            {\@nameuse{long@exp@#2}}%
        \else%
          \ClassError
           {labbook}
           {index entry and experiment title don't match}
           {%
             You have used \protect\experiment\space with an
             optional argument, and used abbreviations
             \MessageBreak both in the optional argument
             (the first item in square brackets, for the index
             and toc\MessageBreak entries) and the mandatory
             argument (in curly braces, for the experiment title
             in the text). This is only possible if both would
             yield the same index/toc\MessageBreak
             entries. However, you requested the index
             entry\MessageBreak
             \@nameuse{short@exp@\fk@current@tocentry}
             \MessageBreak
             but the title corresponds to index entry\MessageBreak
             \@nameuse{short@exp@#2}}%
        \fi%
    \else%
```

This is the working case with a predefined short form and a free long form. We warn the user - she might have accidentaly chosen the abbreviation:

```
      \ClassWarning{labbook}
        {Using a pre-defined short form for this
          \protect\experiment.\MessageBreak
          Please check that the abbreviation\MessageBreak
          \csname short@exp@#1\endcsname\MessageBreak corresponds
          properly to the long form \MessageBreak #2\MessageBreak}
      \@startsection{experiment}{1}{\z@}%
        {-3.5ex \@plus -1ex \@minus -.2ex}%
        {2.3ex \@plus.2ex}%
        {\raggedsection\normalfont\sectfont\nobreak\size@section\nobreak}%
        [\@nameuse{short@exp@\fk@current@tocentry}]{#2}%
    \fi%
  \else%
```

The `\fi` is the end of the conditional regarding the mandatory argument, when the (first element of the) optional argument was an abbreviation. The `\else` case now means that the optional argument is a text, and we have to check again the state of the mandatory argument:

```
    \def\fk@currentarg{#2 }\@onelevel@sanitize{\fk@currentarg}%
    \expandafter\fk@checkifabbrev@arg\fk@currentarg&{long@exp}%
    \ifabbrev@defined%
```

This is the error case with a free optional and a predefined mandatory argument:

```
    \ClassError {labbook} {Manual short form conflicts with
      abbreviated title} {You have used an optional argument to
      \protect\experiment\space (the first element in
      square\MessageBreak brackets) that TeX does not
      recognize as an abbreviation. However, in the
      \MessageBreak experiment title (in the curly braces),
      you have used an abbreviation defined\MessageBreak with
      \protect\newexperiment. This doesn't make sense, so
      I don't accept it.}%
    \else%
```

This is the working case where optional and mandatory argument both are free form:

```
      \@startsection{experiment}{1}{\z@}%
         {-3.5ex \@plus -1ex \@minus -.2ex}%
         {2.3ex \@plus.2ex}%
         {\raggedsection\normalfont\sectfont\nobreak\size@section\nobreak}%
         [\fk@current@tocentry]{#2}%
    \fi%
  \fi%
}%
```

There is no starred form yet - we issue an error. We could as well `\let` it to the unstarred form, but then old documents would change their appearance once a starred form is introduced.

```
\def\@sexperiment#1{%
  \ClassError{labbook}{%
    Starred form of \protect\experiment\space not defined
  }{%
    There is no starred form of \protect\experiment\space defined
  in this version of labbook.cls. Please use the unstarred form, or
  check for a new version.
  }
}
```

8.2 Defining \subexperiment

We define `\subexperiment` analogous to `\experiment`. In order to make `\fk@close@labindex` work properly, we assign explevel and parselevel.

```
\def\subexperiment{%
  \def\fk@explevel{subexp}%
  \def\fk@parselevel{exp}%
  \fk@close@labindex%
  \lower@sectionlevelfalse%
  \@ifstar{\@ssubexperiment}{\@subexperiment}%
}%
```

```
\def\@subexperiment{%
  \@ifnextchar [{\opt@arg@subexperiment}{\nopt@arg@subexperiment}%]
}%
\def\nopt@arg@subexperiment#1{%
  \def\fk@parselevel{exp}
  \def\fk@currentarg{#1 }\@onelevel@sanitize{\fk@currentarg}%
  \expandafter\fk@checkifabbrev@arg\fk@currentarg&{long@subexp}%
  \ifabbrev@defined%
    \fk@buildindexlist{\csname short@subexp@#1\endcsname}%
    \@startsection{subexperiment}{2}{\z@}%
         {-3.5ex \@plus -1ex \@minus -.2ex}%
         {2.3ex \@plus.2ex}%
         {\raggedsection\normalfont\sectfont\nobreak\size@section\nobreak}%
         [\@nameuse{short@subexp@#1}]{\@nameuse{long@subexp@#1}}%
  \else
    \expandafter\fk@buildindexlist{#1}
    \@startsection{subexperiment}{2}{\z@}%
         {-3.5ex \@plus -1ex \@minus -.2ex}%
         {2.3ex \@plus.2ex}%
         {\raggedsection\normalfont\sectfont\nobreak\size@section\nobreak}%
         {#1}%
  \fi%
}%
```

For parsing the optional argument, we need parselevel `subexp`. After that, `\addcontentsline` (which is implicitly called by `\@startsection`) needs parselevel `exp` to properly set the index entries.

```
\def\opt@arg@subexperiment[#1]#2{%
  \def\fk@parselevel{subexp}
  \fk@parse@optarg#1,\relax,%
  \def\fk@parselevel{exp}%
  \ifabbrev@defined%
    \def\fk@currentarg{#2 }\@onelevel@sanitize{\fk@currentarg}%
    \expandafter\fk@checkifabbrev@arg\fk@currentarg&{long@exp}%
    \ifabbrev@defined%
      \expandafter\ifx\csname short@subexp@\fk@current@tocentry%
          \expandafter\endcsname\csname short@subexp@#2\endcsname%
        \@startsection{subexperiment}{2}{\z@}%
           {-3.5ex \@plus -1ex \@minus -.2ex}%
           {2.3ex \@plus.2ex}%
           {\raggedsection\normalfont\sectfont\nobreak\size@section\nobreak}%
           [\@nameuse{short@subexp@\fk@current@tocentry}]%
           {\@nameuse{long@subexp@#2}}%
      \else%
        \ClassError
          {labbook}
          {index entry and subexperiment title don't match}
          {%
            You have used \protect\subexperiment\space with an
            optional argument, and used abbreviations\MessageBreak
```

```
            both in the optional argument (in square brackets, for
            the index and toc\MessageBreak entries) and the
            mandatory argument (in curly braces, for the experiment
            title in the text). This is only possible if both would
            yield the same index/toc\MessageBreak entries. However,
            you requested the index entry\MessageBreak
            \@nameuse{short@subexp@#1}\MessageBreak
            but the title corresponds to index entry\MessageBreak
            \@nameuse{short@subexp@#2}}%
      \fi%
    \else%
       \ClassWarning{labbook}
         {Using a pre-defined short form for this
          \protect\subexperiment.\MessageBreak
          Please check that the abbreviation\MessageBreak \csname
          short@subexp@#1\endcsname\MessageBreak corresponds
          properly to the long form \MessageBreak #2\MessageBreak}%
       \@startsection{subexperiment}{2}{\z@}%
         {-3.5ex \@plus -1ex \@minus -.2ex}%
         {2.3ex \@plus.2ex}%
         {\raggedsection\normalfont\sectfont\nobreak\size@section\nobreak}%
         [\@nameuse{short@subexp@\fk@current@tocentry}]{#2}%
    \fi%
  \else%
    \def\fk@currentarg{#2 }\@onelevel@sanitize{\fk@currentarg}%
    \expandafter\fk@checkifabbrev@arg\fk@currentarg&{long@subexp}%
    \ifabbrev@defined%
      \ClassError
         {labbook}
         {Manual short form conflicts with abbreviated title}
         {You have used an optional argument to
          \protect\subexperiment\space (the short form, in
          \MessageBreak square brackets) that TeX does not
          recognize as an abbreviation. However, \MessageBreak
          in the subexperiment title (in the curly braces),
          you have used an abbreviation\MessageBreak defined
          with \protect\newsubexperiment. This doesn't make
          sense, so I don't accept it.}%
    \else%
      \@startsection{subexperiment}{2}{\z@}%
         {-3.5ex \@plus -1ex \@minus -.2ex}%
         {2.3ex \@plus.2ex}%
         {\raggedsection\normalfont\sectfont\nobreak\size@section\nobreak}%
         [\fk@current@tocentry]{#2}%
    \fi%
  \fi%
}%
\def\@ssubexperiment#1{%
  \ClassError{labbook}{%
    Starred form of \protect\subexperiment\space not defined
```

```
  }{%
    There is no starred form of \protect\subexperiment\space defined
  in this version of labbook.cls. Please use the unstarred form, or
  check for a new version.
  }%
}%
```

8.3 Defining labday

In mainmatter, `\labday` replaces chapter. (`\chapter` may still be used and will get a toc entry, e.g. after `\backmatter`). `\labday` is an extended `\addchap` (i.e. an un-numbered `\chapter` with toc entry and assignment of a running headline) which additionally sets the closing index entry for the preceding `\(sub)experiment`. After closing, `\k@explevel` is made undefined, so that addcontentsline won't try to open index ranges.

```
\newcommand*{\labday}{%
  \def\fk@explevel{exp}%
  \fk@close@labindex%
  \let\fk@explevel\@undefined%
  \refstepcounter{labday}%
  \addchap%
}%
```

8.4 Adjusting the definition of part

If part is used, it must also call `\fk@close@labindex`.

```
\let\fk@part\part
\renewcommand*{\part}{%
  \def\fk@explevel{exp}%
  \fk@close@labindex%
  \let\fk@explevel\@undefined%
  \fk@part%
}
```

8.5 Adjusting subsubsection

If somebody uses lower sectioning levels than `\subexperiment`, the index lists are not affected, and thus `\addcontentsline` would again open the index entries for the last `\subexperiment`. To avoid this, we extend `\subsubsection`; it now sets the conditional `\iflower@sectionlevel` to true. This will be checked by `\addcontentsline`.

```
\newif\iflower@sectionlevel
\let\fk@oldsubsubsection\subsubsection%
\renewcommand{\subsubsection}{%
  \lower@sectionleveltrue%
  \fk@oldsubsubsection%
}
```

8.6 Section numbering, floats and the table of contents

Since the days will not be numbered, we want to also change the numbering scheme for the new sectioning commands:

```
\renewcommand*\theexperiment{\@arabic\c@experiment}%
\renewcommand*\thesubexperiment{%
  \theexperiment.\@arabic\c@subexperiment}%
```

And for consistency also for the lower levels – nobody will ever need these numbered, will one?

```
\renewcommand*\thesubsubsection{%
  \thesubexperiment.\@arabic\c@subsection}%
\renewcommand*\theparagraph{%
  \thesubsubsection.\@arabic\c@paragraph}%
\renewcommand*\thesubparagraph{%
  \theparagraph.\@arabic\c@subparagraph}%
```

And the floats:

```
\@addtoreset{figure}{labday}%
\@addtoreset{table}{labday}%
\renewcommand*\thefigure{%
  \@arabic\c@figure}%
\renewcommand*\thetable{%
  \@arabic\c@table}%
```

To be able to print a table of contents, we have to define `\contentsline` for labday, experiment and subexperiment. We just copy the definitions from `\chapter` and `\(sub)section`:

```
\let\l@labday\l@chapter%
\let\l@experiment\l@section%
\let\l@subexperiment\l@subsection%
```

9 Building the index

9.1 Redefining `addcontentsline`

If `\experiment` or `\subexperiment` are called, then `\addcontentsline` is extended to open the index range entries (closing is done by `\fk@close@labindex`, see above).

Redefining is only done *here* if we do not use `hyperref.sty`. If the `hyperref` option has been specified, it is delayed until after this package has been loaded, see section **??**. What we do here in any case is define a macro that will do the real defining of `\addcontentsline`.

```
%    \begin{macrocode}
\def\define@addcontentsline{%
  \let\fk@old@addcontentsline\addcontentsline%
  \def\addcontentsline##1##2##3{%
```

First, we call all the old commands. After that, we apply the changes, starting with a bunch of checks: First wether we are called from a sectioning command (writing to `toc`), not by a caption of a figure or table; then wether `\fk@explevel` is defined (i.e. wether we are not called by `\part`, `\labday` or a traditional sectioning command outside mainmatter). And last wether we are not called by subsubsection or lower.

```
    \fk@old@addcontentsline{##1}{##2}{##3}%
    \def\@tmpA{toc}%
    \def\@tmpB{##1}%
    \ifx\@tmpA\@tmpB%
      \@ifundefined{fk@explevel}{}{%
        \iflower@sectionlevel\else%
```

We define `\fk@writeindex` to open ranges and then call `\fk@useindexlist` with the experiment's indexlist – as explained above (see **??**), this is also correct for subexperiments.

```
          \let\fk@writeindex\fk@openindex%
          \expandafter\fk@useindexlist\fk@exp@indexlist\relax%
        \fi%
      }%
    \fi%
  }%
}%
```

Now let's see wether hyperref.sty is used, according to the class option. If no, we can redefine `\addcontentsline` right now:

```
%
\ifwe@use@hyperref\else%
  \define@addcontentsline%
\fi%
```

9.2 Making sure the last index range is closed

Since closing of the index entries is usually done by the subsequent call of `\(sub)experiment`, the last would never be closed. The closing has to be done at the begin of the appendix if there is one, or at the end of mainmatter, if `\backmatter` is called, or at last at the end of the document. The user should still be able to call `\appendix` and `\backmatter` in arbitrary order, or not at all. To decide at which place we close, we define a new conditional which is initialized to be false. Then we define the command `\fk@close@labindex` to close the index entries, if one is open (i.e. if `\fk@current@(sub)expname` is not `\relax`). After closing, we `\let` the index names to `\relax`.

The `\appendix` command is extended to call `\fk@close@labindex` if the conditional is still false, so is `\backmatter`. Additionally, after closing the index entries `\fk@explevel` is made undefined, so that subsequent calls of `\addcontentsline` don't even try to use the (albeit empty) index lists.

```
\newif\iflast@labindex@closed\last@labindex@closedfalse%
```

```
\let\fk@old@appendix\appendix%
\def\appendix{%
  \iflast@labindex@closed\else%
    \def\fk@explevel{exp}%
    \fk@close@labindex%
    \last@labindex@closedtrue%
    \let\fk@explevel\@undefined%
    \setcounter{footnote}{0}%
  \fi%
  \fk@old@appendix%
}%
\let\fk@old@backmatter\backmatter%
\def\backmatter{%
  \iflast@labindex@closed\else%
    \def\fk@explevel{exp}%
    \fk@close@labindex%
    \last@labindex@closedtrue%
    \let\fk@explevel\@undefined
    \setcounter{footnote}{0}%
  \fi%
  \fk@old@backmatter%
}
```

And as a last resort, `\AtEndDocument` will do it.

```
\AtEndDocument{%
  \iflast@labindex@closed\else%
    \def\fk@explevel{exp}%
    \fk@close@labindex%
  \fi%
}
```

10 hyperref compatibility macros

First we redefine `\addcontentsline` which was delayed, the definition is explained above (see **??**)

```
\ifwe@use@hyperref
  \AfterPackage{hyperref}{%
    \define@addcontentsline%
    \providecommand*{\toclevel@labday}{0}%
    \providecommand*{\toclevel@experiment}{1}%
    \providecommand*{\toclevel@subexperiment}{2}%
    \newcommand*\theHlabday{%
      \arabic{labday}}%
    \newcommand*\theHexperiment{%
      \theHlabday.\arabic{experiment}}%
    \newcommand*\theHsubexperiment{%
      \theHexperiment.\arabic{subexperiment}}%
    \renewcommand*\theHsubsubsection{%
```

```
      \theHsubexperiment.\arabic{subsubsection}}%
    \renewcommand*\theHfigure{%
      \theHlabday.\arabic{figure}}%
    \renewcommand*\theHtable{%
      \theHlabday.\arabic{table}}%
    \newcommand*\theHsubfigure{%
      \theHfigure.\arabic{subfigure}}
  }%
  \AtBeginDocument{%
    \@ifpackageloaded{hyperref}{}{%
      \ClassError{labbook}{%
        hyperref option given, but package not loaded}{%
        You have specified the class option hyperref, but
        not loaded the package until \protect\begin{document}.
      }
    }
  }
\else
  \AtBeginDocument{%
    \@ifpackageloaded{hyperref}{%
      \ClassError{labbook}{%
        hyperref option not given, but package loaded}{%
        You have not specified the class option hyperref,
        but loaded the package. Don't do that again!
      }%
    }{}%
  }%
\fi%
```
⟨/labbook⟩