Documentation of the LaTeX packages

sets and vhistory

Jochen Wertenauer

jwertenauer@gmx.de

July 21, 2007

Contents

1 Introduction

It's clear to me that nobody reads introductions. Nevertheless I recommend not to skip this section, as it tries to explain, why I developed the packages sets and vhistory. By reading the following you can see, if these packages meet your requirements.

During software projects (hopefully) a lot of documents like specification or design are created. These are normally revised several times. To reconstruct changes, documents should contain a so called *history of versions* aka change log. This is a table, whose entries contain the following data:

- a version number,
- the date of the change,
- some kind of identification code (e. g. the initials) of the persons, that made the changes (the authors),
- a summary of the changes.

Certain data of the history of versions shall often be repeated at other parts of the document. Typically the title-page shall quote the most recent version number and all authors of the document. The version number should although be repeated on every page, e. g. in a footer. By doing this, it is easy to verify, if a page is part of the most recent version or if it is outdated.

Normally the data, that is placed on say the title-page, is not (automatically) taken from the history of versions, but stated at an other part of the document. The entries in the history of versions are normally always updated. Due to stress and rush updating the data on the title-page is often forgotten. This results in inconsistent documents. From my own experience I know, that the list of authors is practically never correct, especially when several people edited a document in the course of time.

Therefore it would be nice, if the author of a document only had to take care of the history of versions. The informations on title-page and in footers should be generated from the history of versions automatically.

These requirements cannot be implemented without any effort because the titlepage for instance is processed by LaTeX before the history of versions is even read. All relevant data therefore has to be written to a file and read again before processing the titlepage. As for some applications even the moment when the aux-file is read is too late, an own file with the extension hst is created. The table containing the history of versions needs an own file, too, but this topic will be discussed later.

Another problem is the list of authors. This list cannot be created be simply concatenating the author entries because this would result in duplicate entries. At this time the package sets was born allowing to manage simple sets of text. At every entry in the history of versions, the set of authors is merged with the set of authors given at this entry. The alphabetically sorted set of authors can then be printed wherever one likes—normally on the tile-page.

The following to sections describe the two packages in more detail and explain, how to use them. I decided not to repeat anything of source code here. Those who are interested can take a look at the sources. I tried to structure and comment the sources in a way that makes it readable for humans.

2 The package vhistory

The purpose of this package was described in detail in section 1. I told you, it is worth reading, didn't I? In this section the usage of vhistory is described in detail.

2.1 Loading the package

As usual the package is loaded by using the command

`\usepackage{vhistory}`

in the preamble. Vhistory requires LaTeX 2ε and the packages sets and ltxtable (which requires longtable and tabularx). If this packages are not already loaded, vhistory will load them.

Vhistory understands several options to control its behaviour. These are listed below. An example for using options would be

`\usepackage[tocentry, owncaptions]{vhistory}`.

nochapter: If this option is used, the history of versions will no be placed in an own chapter—respectively an own section, if you use the document classes article or scrartcl.

tocentry: With this option you can cause an entry in the table of contents. Normaly there won't be such an entry. If the option nochapter is used, tocentry has no effect and you are responsible for toc entries.

owncaptions: vhistory supports the languages English and German. If you use an unsopported language, all captions (like "History of versions" or "Changes") will be in English. In this case you might want to change captions yourself. The option owncaptions supports you in this case. More information on this topic can be found in subsection 2.3.

hideauthorcolumn: If the document to create will only have one author, the column "autor(s)" in the history of versions is maybe unwanted. By using this option, you suppress the display of the column. The syntax of all other commands remains unchanged. In particular it is still possible to print a list of all authors.

2.2 Usage

The Usage of vhistory is rather easy and is to be described in this subsection. It is important to know that vhistory needs to runs of LaTeX because some data is written to external files.

2.2.1 Creating the history of versions

The history of versions is an environment:

```
\begin{versionhistory}
<entries>
\begin{versionhistory}
```

An entry generally looks like:

```
\vhEntry{<Version>}{<Date>}{<Authors>}{<Changes>}
```

The authors are written in the set notation of the package sets, i. e. the character `|` is used as the separator. An example for entry could therefore be:

```
\vhEntry{1.1}{13.05.04}{JW|AK|KL}{Typos corrected.}
```

The `\begin`... starts a new chapter (or a new section respectively, if article is used), if not disabled as described above.

The history of versions itself is set in an "ltxtable". As a result of that, the history of versions can consist of more than one page. Lines in the columns "Author(s)" and "Changes" are broken automatically. The package ltxtable requires the table to be in a separate file. This file is generated be vhistory an has the ending "ver".

2.2.2 Getting the current version number

The current version number—precisely: the version number of the last entry—can be read with the command

```
\vhCurrentVersion
```

.
The command is available right after loading the package.

2.2.3 Getting the current date

Analogous to the current version number, the date of the last change can be printed. By using the command

```
\vhCurrentDate
```

you will get this data. The command is available right after loading the package, too.

2.2.4 Putting a list of all authors

You can get a list of all authors on two ways. Via the command

`\vhAllAuthorsSet`

you can get the authors as a set as described in section 3. The more convenient way is using the command

`\vhListAllAuthors`.

This command returns an alphabetically sorted list of authors. The items of the list are comma separated. If you want another separator—e. g. & for usage in a table—to be used instead, you can redefine the command

`\setseparator`

from the package sets (compare section 3.1.2).

Sometimes you want the list to contain full names like "Jochen Wertenauer" instead of the short form "JW", which would probably be used in the history of versions. By using the command

`\vhListAllAuthorsLong`

you can get this behaviour. In this case you still write "JW" in the `\entry` command (note: there is no backslash here!), but also define a macro `\JW` like shown below.

```
\newcommand{\JW}{Jochen Wertenauer}
```

In the history of versions, the text "JW" will be shown in the authors column, but `\vhListAllAuthorsLong` will use the macro `\JW`. If the macro is undefined, no text will be shown (for this author).

2.3 Language support

As mentioned is section 2.1, vhistory supports the languages English and German. All language dependent captions with their defaults are listed in table 1. If the document is written in a unsopported language, the English version will be used.

By providing the option "owncaptions" you can use custom captions. This option causes vhistory to default the commands in the language that was selected, when vhistory is loaded. Therefore it makes sense to load packages like babel or ngerman before vhistory.

A caption can be changed via the command

`\renewcommand{<Command>}{<custom text>}`.

If you prefer to use "Improvements" instead of "Canges", you have to write

```
\renewcommand{\vhchangename}{Improvements}
```

to get the desired result.

Command	German	English
\vhhistoryname	Versionshistorie	History of Versions
\vhversionname	Version	Version
\vhdatename	Datum	Date
\vhauthorname	Autor(en)	Author(s)
\vhchangename	Änderungen	Changes

Table 1: language dependent captions

2.4 Example

```
\documentclass{scrartcl}
\usepackage{vhistory, hyperref}

\newcommand{\docTitle}{An example for vhistory}

\hypersetup{%
  pdftitle  = {\docTitle},
  pdfkeywords = {\docTitle, Version \vhCurrentVersion
                 from \vhCurrentDate},
  pdfauthor = {\vhAllAuthorsSet}
}

\usepackage{scrpage2}
\pagestyle{scrheadings}
\ifoot{\docTitle\ -- Version \vhCurrentVersion}
\cfoot[]{}
\ofoot[\thepage]{\thepage}

\begin{document}

\title{\docTitle}
\author{\vhListAllAuthors}
\date{Version \vhCurrentVersion\ from \vhCurrentDate}
\maketitle

\begin{versionhistory}
  \vhEntry{1.0}{22.01.04}{JPW|KW}{created}
  \vhEntry{1.1}{23.01.04}{DP|JPW}{correction}
  \vhEntry{1.2}{03.02.04}{DP|JPW}{revised after review}
\end{versionhistory}

\end{document}
```

Figure 1: Example for the usage of vhistory

3 The package sets

As described in the introduction, sets was designed to support set operations. The elements of a set are normally simple Text, but you can insert commands in a set, too. These commands will—except when printing the set—not expanded. The usage of braces ('{' and '}') unfortunately doesn't work. In this case you have to define a shortcut that doesn't need braces. However, Parameters without braces work. "`H"agar`" is therefore a valid element of a set. "`\endset`" and "`\empty`" can not be part of a set.

As a document has only a few authors, not much effort was put in improving efficiency. Sets should therefore be relatively small. If you nevertheless try to create a set with hundreds or thousands of items, TEX's stack might overflow.

In most cases the sequence of items doesn't matter. This is also the case in most commands here. Exceptions will be marked.

The package sets needs LATEX 2_ε.

3.1 Usage

In this subsection the usage of the package sets is described. In the description, some example sets will be used:

$$\begin{aligned} A &= \{Alice, Bob, Charly\} \\ B &= \{Alice, Bob\} \\ C &= \{Bob, Dean\} \\ D &= \{Dean\} \\ L &= \emptyset \end{aligned}$$

3.1.1 Constructors

To create a set, the commands

```
\newset{<set>}{<content>}
```

and

```
\newsetsimple{<set>}{<content>}
```

can be used. `<set>` is a command name to access the set from now on. The items of a set are separated with `|`. The set A could therefore be defined with:

```
\newset{\sA}{Alice|Bob|Charly}
```

The set L is defined with:

```
\newset{\sL}{}
```

`\newset` creates a new set. This set will be sorted in alphabetical order and duplicates will be removed. So it would be no matter, if in the definition of A after "Charly" a second "Alice" was inserted.

The effort for sorting and duplicate deleting is unnecessary at this point. If you want to skip these expensive steps, you can create a set with the command `\newsetsimple`, too.

Because they are needed later, we will create all sets mentioned above:

```
\newsetsimple{\sA}{Alice|Bob|Charly}
\newsetsimple{\sD}{Alice|Bob}
\newsetsimple{\sC}{Bob|Dean}
\newsetsimple{\sD}{Dean}
\newsetsimple{\sL}{}
```

3.1.2 Inspectors

Inspectores help you to retrieve informations about sets and to print sets.

Printing: A set can be printed using the command

`\listset`.

The elements will be put in the sequence they are in the set. A comma is used as separator.

`\listset{\sA}` therefore leads to the following output:

Alice, Bob, Charly

Sometimes you might want to separate the items in a different way, for example with a `&` to put them in a table. In this case a (temporary) redefinition of

`\setseparator`

helps you. Normally this command expands to '`,\␣`'.

Determining the size of a set: The next inspector has the syntax

`\sizeofset{`S`}\is{<counter>}`,

where `<counter>` is the name of a LaTeX counter, which afterwards will contain the number of elements in set S. The sequence

```
\newcounter{mycounter}
\sizeofset{\sB}\is{mycounter}
\arabic{mycounter}
```

leads to the output: "2" If you determine the size of set L, the result—as you might have expected—is "0".

Testing for membership: By using the command

`\iselementofset{`e`}{`S`}`

you can check, whether $e \in S$ is true. The effort is $O(1)$ because all work is done by the pattern matching of TEX. The sequence

```
\if \iselementofset{Bob}{\sC}Yes\else No\fi
```

would result in the output "Yes", the same test with set D in "No".

3.1.3 Modificators

Union of sets: The operation $R := S_1 \cup S_2$ is realized in the command

`\unionsets{`S_1`}{`S_2`}\to{`R`}`.

Table 2 contains some examples. The result of the operation is a sorted set without duplicates containing the items of sets S_1 and S_2.

Difference of sets: The operation $R := S_1 - S_2$ (also written as $R := S_1 \backslash S_2$) can be carried out with

`\minussets{`S_1`}\minus{`S_2`}\to{`R`}`.

If S_1 is a sorted set, R will be sorted, too. If S_1 contains duplicates, R might also contains these duplicate elements. Table 2 contains several examples for the usage of this command.

Colloquially you can formulate the operation as follows: Check for every element e in S_1, if $e \in S_2$ is true. If not, insert e into R. And that's exactly the way it has been implemented!

Intersection of sets: The operation $R := S_1 \cap S_2$ is made possible with the command

`\intersectsets{`S_1`}{`S_2`}\to{`R`}`.

As above: If S_1 is a sorted set, R will be sorted, too. If S_1 contains duplicates, R might also contains these duplicate elements. Table 2 contains several examples for the usage of this command, too.

This operation can colloquially be written down as: Check for every element e in S_1, if $e \in S_2$ is true. If yes, insert e into R. If you compare this with the formulation above, one can recognize that the only difference is the small word "yes". In the source code, this expresses in a missing `\else`. Acutally amazingly simple, if you remember the formal relation $S_1 \cap S_2 \equiv S_1 \backslash (S_1 \backslash S_2)$, which lets one expect a much higher complexity.

Sorting: A set S can be sorted alphabetically by using the command

`\sortset{`S`}{`R`}`.

Operation	Result
`\unionsets{\sA}{\sC}\to{\sR}`	"Alice, Bob, Charly, Dean"
`\unionsets{\sB}{\sD}\to{\sR}`	"Alice, Bob, Dean"
`\unionsets{\sL}{\sC}\to{\sR}`	"Bob, Dean"
`\unionsets{\sL}{\sL}\to{\sR}`	""
`\minussets{\sA}\minus{\sC}\to{\sR}`	"Alice, Charly"
`\minussets{\sD}\minus{\sC}\to{\sR}`	""
`\minussets{\sD}\minus{\sB}\to{\sR}`	"Dean"
`\minussets{\sA}\minus{\sL}\to{\sR}`	"Alice, Bob, Charly"
`\intersectsets{\sA}{\sB}\to{\sR}`	"Alice, Bob"
`\intersectsets{\sC}{\sB}\to{\sR}`	"Bob"
`\intersectsets{\sB}{\sD}\to{\sR}`	""
`\intersectsets{\sA}{\sL}\to{\sR}`	""

Table 2: Set operations, examples

After execution of the command, *R* contains the sorted set. The sorting is done by the bubblesort algorithm, an algorithm, that can be implemented in TEX without having to perform many contortions.

At sorting, the elements are compared as they are, i. e. possibly contained macros are not expanded but compared by their name (including the backslash).

Removing duplicates: The operations *only* works on sorted sets! You actually will not need it very often because the creation of a set using `\newset` does the work automatically (by using this macro). However, I decided to make this macro available to public; probably sometimes someone really needs it. Duplicate removal is called with:

`\deleteduplicates{`*S*`}{`*R*`}`,

with *R* being the result set and *S* being the sorted set, whose duplicates shall be removed.

3.2 Effort estimations

Table 3 lists the complexity of the operations in O-notation. The following Assumptions are made:

- Let the length of an element of a set be m.
- Let the number of elements in a set be n. If an operation need two sets, n_1 is the cardinality of the first set and n_2 the cardinality of the second one.

- For simplicity reasons the effort for pattern matching in parameter processing is assumed to be constant.

The given complexity classes can help you to arrange a set of operations in the optimal sequence. For example when using the commands `\intersectsets` or `\minussets` it is better to let the smaller set be the first parameter.

Operation	**Operation**
Compare elements	m
`\sizeofset`	n
`\listset`	n
`\iselementofset`	1
`\sortset`	$m \cdot n^2$
`\deleteduplicates`	n
`\newset`	$m \cdot n^2$
`\newsetsimple`	1
`\unionsets`	$m \cdot (n_1 + n_2)^2$
`\intersectsets`	n_1
`\minussets`	n_1

Table 3: Complexity classes of set operations