Das dateiliste-Package – Liste der verwendeten Dateien im Dokument*

Paul Ebermann[†‡]

13. Oktober 2012

Zusammenfassung

Dieses Paket implementiert einige Befehle, um den Überblick über Versionen und Änderungsdaten von LaTeX-Quelltexten zu behalten: Automatische Versionsinfos aus dem Versionskontrollsystem (CVS/RCS oder Subversion), Hauptdatei in der Dateiliste, Inklusion der Dateiliste als Tabelle im Dokument.

Inhaltsverzeichnis

*Dieses Dokument gehört zu dateiliste v0.6, vom 2012/10/13.

†E-Mail: Paul-Ebermann@gmx.de

‡Rolf Niepraschk (Rolf.Niepraschk@ptb.de) hat das Package printfilelist geschrieben und mir geschickt, dessen Code bildete die Basis für \printFileList. Für den jetztigen Code (insbesondere dessen Fehler) bin ich (Paul) aber selbst verantwortlich.

1 Benutzerdokumentation der Befehle

Die drei Teile *automatische Versionsinfos* (mittels `\ProvideFileInfos`), *Hauptdatei in der Dateiliste* (mittels `\mainFileToList` bzw. der `addmain`-Package-Option) und *Dateilisten-Ausgabe* (mittels `printFileList`) sind unabhängig voneinander nutzbar, aber die ersten beiden Features werden durch das letzte erst richtig nützlich, und damit das letzte eine vernünftige Ausgabe hat, sind die ersten beiden hilfreich. Daher also alles in einem Package.

1.1 README in Englisch, Deutsch und Esperanto

Eine README-Datei mit einer Kurzübersicht des Paketes auf Englisch, Deutsch und Esperanto (sowie einzel-Dateien für diese Sprachen) kann ebenfalls mit docstrip aus der dtx-Datei generiert werden, die beiliegende Datei `dateiliste.ins` tut dies neben der Generierung der `.sty`-Datei.

1.2 Automatische Versionsinfos

`\ProvideFileInfos` `{`⟨*id-string*⟩`}{`⟨*kurzbeschreibung*⟩`}`

Ändert die Informationen für die Datei, in der es aufgerufen wurde (bzw. fügt neue hinzu).

⟨*kurzbeschreibung*⟩ sollte eine kurze Beschreibung der Funktion/des Inhaltes der Datei sein, optimalerweise nur ASCII-Zeichen.

⟨*id-string*⟩ sollte ein String sein, der (im RCS-Modus) wie

```
$Id: dateiliste.dtx,v 4.1 2009/10/31 19:58:13 epaul Exp $
```

oder (im Subversion-Modus) wie

```
$Id$
```

aussieht. Diesen lässt man am besten von seinem Versionskontrollsystem produzieren – dieses Paket unterstützt die Syntax von RCS und CVS (RCS-Modus) sowie die Syntax von Subversion (svn).

Der Modus kann durch Paket-Optionen ausgewählt werden:

rcs
cvs

- Die Option `rcs` oder `cvs` wählt den RCS-Modus aus, in dem das Datum mit `/` getrennt wird, der Dateiname von `,v` gefolgt wird, am Ende ein Status gezeigt wird (meist `Exp`), und die Versionsnommer eine mit `.` getrennte Zahlenfolge ist. (Es kann außerdem noch der Nutzername desjenigen enthalten sein, der eine Sperre auf diese Datei hält.)

subversion
svn

- Die Option `svn` oder `subversion` wählt den Subversion-Modus aus, in dem der Dateiname ohne Zusatz auftaucht, Datum und Uhrzeit im ISO-8601-Format mit Zeitzonenanzeiger `Z` dargestellt werden, und die Versionsnummer nur eine Zahl ist.

Aus Kompatibilitätsgründen zu früheren Versionen dieses Paketes (vor 0.5) ist der Default-Modus (d.h. wenn keine dieser Optionen angegeben wurde) `rcs`.

Wir schreiben also etwas wie

```
\ProvideFileInfos{$Id$}{Algebraische Geometrie I}
```

in unsere Datei (üblicherweise ziemlich am Anfang) und rufen dann `cvs commit` bzw. `svn commit` auf – dadurch werden von CVS bzw. Subversion die passenden Informationen in unsere Arbeitskopie eingefüllt. (Bei Subversion muss vorher die Keyword-Ersetzung eingeschaltet werden, etwa mit

`svn propset svn:keywords Id` ⟨*dateiname*⟩,

bei CVS muss man nur darauf achten, sie nicht abzuschalten.) Analoges geschieht auch beim Checkout.

Das Makro sorgt dann dafür, dass in der durch `\listfiles` sowie auch der durch `\printFileList` (siehe Abschnitt 1.4) erzeugten Liste die richtigen Daten (d.h. das Commit-Datum und die RCS/CVS/SVN-Versionsnummer) stehen.

Dies ist besonders nützlich, wenn man ein Dokument hat, welches sich aus mehreren Quelltextdateien zusammensetzt, welche üblicherweise dann auch nicht immer gleichzeitig bearbeitet werden.

1.3 Hauptdatei in der Dateiliste

`\mainFileToList` Fügt die Haupt-Datei (⟨*jobname*⟩`.tex`) am Anfang der Dateiliste ein, falls sie existiert. Zuvor wird versucht, ein etwaiges Vorkommen dieses Namens in der Liste aus dieser zu entfernen. (Ein solches Vorkommen kann nur dann sein, wenn die Haupt-Datei mit `\input{...}` geladen wurde, anstatt einfach so den Namen auf der Kommandozeile anzugeben, oder wenn die Hauptdatei sich selbst noch einmal einliest, wie in diesem Dokument.)

`cat12` `nocat12` Die interne Funktionsweise (und damit etwaige Nebenwirkungen) hängt davon ab, ob eine der Package-Optionen `cat12` oder `nocat12` gegeben wurde:

- Mit Option `cat12` wird die interne Liste der geladenen Dateien in Kategorie-12-Zeichen (d.h. other) umgewandelt, um den eventuell dort schon vorhandenen Namen der Hauptdatei zu löschen. Dies kann Nebenwirkungen auf andere Pakete haben, wenn diese ebenfalls diese Liste verarbeiten und dabei auf die Kategorien der Zeichen angewiesen sind.

- Mit Option `nocat12` funktioniert es bei Verwendung von ε-TEX auch ohne die eben geschilderte Umwandlung, bei TEX ist das nicht möglich und es wird einfach so der Dateiname eingefügt – mit dem Effekt, dass er eventuell doppelt auftaucht.

- Ohne diese Optionen wird bei Verwendung von ε-TEX die nebenwirkungsfreie Variante, ohne ε-TEX die Kategorie-12-Variante gewählt.

Details dazu sind im Implementationsteil nachzulesen.

`noaddmain` `addmain`

Dieses Makro wird automatisch am Ende des Dokumentes ausgeführt, falls nicht die Package-Option `noaddmain` gesetzt wurde. Mit `addmain` kann das Vorgabe-Verhalten erzwungen werden.

1.4 Dateilisten-Ausgabe

`\printFileList` `[`⟨*gliederung*⟩`]`

Fügt an der aktuellen Stelle eine Liste der im aktuellen Dokument verwendeten Dateien (ohne die Haupt-Datei) ein. ⟨*gliederung*⟩ ist ein Gliederungsbefehl (wie `\section`, `\chapter` etc., Vorgabewert ist `\section*` (für einen unnummerierten Abschnitt).

Damit eine Liste ausgegeben wird, muss in der Präambel des Dokumentes ein `\listfiles` auftauchen. (Andernfalls gibt `\printFileList` nur eine Warnung auf der Konsole aus und tut sonst nichts.)

Die Liste selbst ist erst ab dem zweiten LATEX-Lauf im Dokument zu sehen (und enthält ab dem dritten Lauf dann auch die Dateilisten-Datei).

`showpages` `noshowpages`

Die Liste wird als 4- oder 5-spaltige Tabelle gesetzt: Falls die Option `showpages` übergeben wurde, ist auch die Seitennummer, die zum Ladezeitpunkt der Datei aktuell war, enthalten (0 für alle Dateien, die in der Präambel geladen wurden), mit `noshowpages` fehlt diese Spalte. (`showpages` ist der Vorgabewert.)

Falls das Paket babel vor oder nach diesem Paket geladen wird, sind die Spaltenüberschriften, die Gliederung sowie die Präambel auch übersetzbar – zur Zeit werden von diesem Paket die Sprachoptionen `english`, `german`, `ngerman` und `esperanto` unterstützt. Wer weitere Übersetzungen beisteuern will, melde sich bei mir.

`\fileListName` `\fileListPreamble` `\fileNameName` `\dateName` `\verName` `\descriptionName`

Ansonsten kann durch Neudefinition der Befehle `\fileListName` (Überschrift), `\fileListPreamble` (einleitender Text, normalerweise mit Fußnote), sowie `\fileNameName`, `\dateName`, `\verName` und `\descriptionName` (Tabellenkopf) der Inhalt der statischen Texte verändert werden. (Auch das wirkt sich erst im folgenden LATEX-Lauf aus.)

Im folgenden mal ein Beispiel aus diesem Dokument, erzeugt mit:

```
\printFileList[\subsubsection]
```

1.4.1 Liste der Dateinamen

Hier die Liste aller Dateien, die während des LaTeX-Laufes, welcher dieses Dokument erzeugte, verwendet wurden. [1]

Dateiname	Seite	Datum	Ver.	Beschreibung
ltxdoc.cls	0	2007/11/11	v2.0u	Standard LaTeX documentation class
article.cls	0	2007/10/19	v1.4h	Standard LaTeX document class
size10.clo	0	2007/10/19	v1.4h	Standard LaTeX file (size option)
doc.sty	0	2010/02/04	v2.1e	Standard LaTeX documentation package (FMi)
multicol.sty	0	2011/06/27	v1.7a	multicolumn formatting (FMi)
pauldoc.sty	0	2009/11/06	v0.5	Pauls Anpassungen fuer doc (PE)
inputenc.sty	0	2008/03/30	v1.1d	Input encoding file
utf8.def	0	2008/04/05	v1.1m	UTF-8 support for inputenc
t1enc.dfu	0	2008/04/05	v1.1m	UTF-8 support for inputenc
ot1enc.dfu	0	2008/04/05	v1.1m	UTF-8 support for inputenc
omsenc.dfu	0	2008/04/05	v1.1m	UTF-8 support for inputenc
babel.sty	0	2008/07/08	v3.8m	The Babel package
ngermanb.ldf	0	2008/07/06	v2.6n	new German support from the babel system
fontenc.sty	0	—		
t1enc.def	0	2005/09/27	v1.99g	Standard LaTeX file
dox.sty	0	2009/09/28	v2.1	Extensions to the doc package
kvoptions.sty	0	2009/12/08	v3.6	Keyval support for LaTeX options (HO)
keyval.sty	0	1999/03/16	v1.13	key=value parser (DPC)
kvsetkeys.sty	0	2010/01/28	v1.8	Key value parser (HO)
infwarerr.sty	0	2007/09/09	v1.2	Providing info/warning/message (HO)
etexcmds.sty	0	2010/01/28	v1.3	Prefix for e-TeX command names (HO)

[1] genauer: Es ist die Liste aller Dokumente, die einen LaTeX-Lauf früher verwendet wurden. Aber nach einigen Läufen sollte sich die Liste stabilisieren.

Dateiname	Seite	Datum	Ver.	Beschreibung
dateiliste.sty	0	2012/10/13	v0.6	Ausgabe der Dateiliste (PE)
svninfo.sty	0	2010/03/22	v0.7.4	
svninfo.cfg	0	—		
ifthen.sty	0	2001/05/26	v1.1c	Standard LaTeX ifthen package (DPC)
ltxtable.sty	0	1995/12/11	v0.2	longtable/tabularx merge (DPC)
tabularx.sty	0	1999/01/07	v2.07	'tabularx' package (DPC)
array.sty	0	2008/09/09	v2.4c	Tabular extension package (FMi)
longtable.sty	0	2004/02/01	v4.11	Multi-page Table package (DPC)
enumerate.sty	0	1999/03/05	v3.00	enumerate extensions (DPC)
test_datei.tex	1	2006/09/19	v0.0	Eine %-$-Beisˆp_iel-3-&-Datei #$ mit {Test} und noch einer Gruppe.
dateiliste.dtx	1	—		
t1cmss.fd	1	1999/05/25	v2.5h	Standard LaTeX font definitions
t1cmtt.fd	1	1999/05/25	v2.5h	Standard LaTeX font definitions
omscmr.fd	2	1999/05/25	v2.5h	Standard LaTeX font definitions
dateiliste.filelist	6	2012/10/13	—	automatically generated filelist
dateiliste.gls	26	—		
dateiliste.ind	27	—		

1.5 Abhängigkeiten

Für die Funktion des Paketes sind die Pakete rcsinfo (Jürgen Vollmer) im RCS-Modus bzw. svninfo (Achim D. Bruckner) im Subversion-Modus und ltxtable – damit auch tabularx, longtable (alle drei von David Carlisle) und array (Frank Mittelbach) – notwendig. babel (Johannes Braams) wird, falls ebenfalls geladen, auch genutzt.

Falls das Paket pauldoc (von mir) ebenfalls geladen wird, werden einige spezielle Anpassungen getroffen.

Für das Setzen der Doku ist außerdem pauldoc (von mir) und dox (von Didier Verna) notwendig.

Für die korrekte Erkennung, dass der Name der Hauptdatei schon in der Liste der geladenen Dateien auftaucht, ist der primitive ε-TEX-Befehl `\scantokens` notwendig – falls kein ε-TEX verwendet wird, kann eine nicht-Erkennung (und damit

am Ende das doppelte Auftauchen der Haupt-Datei) vorkommen. Details dazu bei der Diskussion der Optionen `cat12` und `nocat12`, sowie im Implementations-Teil.

1.6 Wunschliste

Ein paar Ideen, die ich gerne mal umsetzen möchte, aber noch nicht dazu gekommen bin:

- eine Option, um die Datumsdarstellung in der Dateiliste auf das ISO-Format (YYYY-MM-DD) umzustellen.
- eine Option, um das `v` in den Versionsnummern wegzulassen.
- eine Möglichkeit, doppelt geladene Dateien in der Liste nicht doppelt aufzuführen. (Das kann etwa passieren, wenn man mitten im Dokument die Eingabe-Kodierung wechselt.)
- eine Möglichkeit, die Liste zu filtern, und z.B. nur `.tex`-Dateien aufzunehmen. Das könnte z.B. nützlich sein, wenn man nur die „Inhalts-Dateien“ eines zusammengesetzen Dokumentes aufführen will, nicht aber alle verwendeten Package-Dateien.
- Falls eine Versionsnummer fehlt, sollte das erkannt werden, und die Spalte leergelassen werden, anstatt den Beschreibungs-Text in die Versionsspalte zu setzen.

2 Implementation

```
100 %<*package>
```

2.1 Optionen

addmain
noaddmain
\if@dateiliste@addMain

Wir merken uns die ausgewählte Option in einem `\if`.

```
101 \newif\if@dateiliste@addMain
102 \DeclareOption{addmain} {%
103    \@dateiliste@addMaintrue
104 }
105 \DeclareOption{noaddmain} {%
106    \@dateiliste@addMainfalse
107 }
```

cat12
nocat12
\if@dateiliste@catxii
\if@dateiliste@nocatxii

Das gleiche gilt für `cat12`/`nocat12` – allerdings haben wir hier zwei `\if`s, da es auch noch die Möglichkeit gibt, keine der beiden Optionen auszuwählen.

```
108 \newif\if@dateiliste@catxii \@dateiliste@catxiifalse
109 \newif\if@dateiliste@nocatxii \@dateiliste@nocatxiifalse
110 \DeclareOption{cat12} {%
111    \@dateiliste@catxiitrue
112 }
113 \DeclareOption{nocat12} {%
```

```
    \@dateiliste@nocatxiitrue
}
```

showpages
noshowpages

Eine weiteres Optionspaar betrifft die Anzeige der Seitennummern in der Dateiliste.

\if@dateiliste@showpages

```
\newif\if@dateiliste@showpages \@dateiliste@showpagesfalse
\DeclareOption{showpages} {%
  \@dateiliste@showpagestrue
}
\DeclareOption{noshowpages} {%
  \@dateiliste@showpagesfalse
}
```

svn
subversion
rcs
cvs

Das vierte Optionspaar: `svn` (mit Alias `subversion`) für Subversion-Ids, `rcs` (mit Alias `cvs`) für CVS/RCS-Ids. Diese unterscheiden sich vor allem im Format der Datumsangaben. (Hier merken wir uns nur die Optionen, die Arbeit geschieht in Abschnitt 2.3.

\if@dateiliste@subversion

```
\newif\if@dateiliste@subversion \@dateiliste@subversionfalse
\DeclareOption{svn} {%
  \@dateiliste@subversiontrue
}
\DeclareOption{subversion} {%
  \@dateiliste@subversiontrue
}
\DeclareOption{cvs} {%
  \@dateiliste@subversionfalse
}
\DeclareOption{rcs} {%
  \@dateiliste@subversionfalse
}
```

Die Standard-Optionen sind `addmain`, `showpages` und `rcs`.

```
\ExecuteOptions{addmain}
\ExecuteOptions{showpages}
\ExecuteOptions{rcs}
\ProcessOptions
```

2.2 Seitenzahlen in die Dateiliste

\@addtofilelist

Falls nicht die Option `noshowpages` angegeben wurde, merken wir uns die Seitennummern aller eingebundenen Dateien. Dazu definieren wir `\@addtofilelist` neu, um neben dem Hinzufügen des Namens zur Dateiliste auch die jeweilige Seitennummer in einem eigenen Makro zu merken.

```
  \if@dateiliste@showpages
    \CheckCommand*{\@addtofilelist}[1]{\xdef\@filelist{\@filelist,#1}}
    \renewcommand*{\@addtofilelist}[1]{%
      \expandafter\xdef%
```

```
        \csname dateiliste@page@#1\endcsname%
        {\thepage}%
        \xdef\@filelist{\@filelist,#1}%
    }
    \fi
```

Damit die Seitennummern für in der Präambel geladene Pakete etc. alle 0 sind, setzen wir die Seitenzahl jetzt auf 0, und bei Beginn des Dokumentes wieder auf 1.

```
\setcounter{page}{0}
\AtBeginDocument{\stepcounter{page}}
```

2.3 Aktuelle Versionsnummern in der Dateiliste

Wir wollen in der durch `\ProvideFileInfo` provozierten Ausgabe automatisch sinnvolle Infos haben, mit Daten, die von unserem Versionskontrollsystem eingefüllt werden. Zur Zeit unterstützen wir das Format von CVS und RCS (schon seit Version 0.0) und das Format von Subversion (seit Version 0.5).

2.3.1 Subversion-Modus

Die Fallunterscheidung funktioniert mit diesem `\if@...`, welches durch Paket-Optionen initialisiert wurde.

```
\if@dateiliste@subversion
```

Im Subversion-Modus laden wir svninfo.

Wir verwenden die Parameter `nofancy`, weil sonst die Fußzeile umgestellt wird, und `notoday`, weil sonst das aktuelle Datum umgestellt wird. `final` verhindert, dass die Fußzeile geändert wird (wir brauchen die Infos ja nur für unsere Dateiliste).

```
\RequirePackage[nofancy, notoday, final]{svninfo}[2006/05/11]
```

`\ProvideFileInfos` `{`⟨*id-string*⟩`}{`⟨*kurzbeschreibung*⟩`}`

```
\newcommand*{\ProvideFileInfos}[2] {%
```

Zunächst lassen wir `\svnInfo` den ⟨*id-string*⟩ analysieren. Dies definiert (unter anderem) die Makros `\svnInfoFile` (der Dateiname), `\svnInfoDay`, `\svnInfoMonth` und `\svnInfoYear` (Datum) und `\svnInfoRevision` (die Revisionsnummer, in der die Datei das letzte mal geändert wurde).

Das Leerzeichen nach dem `#1` ist notwendig, damit `\svnInfo` erkennt, wo der ⟨*id-string*⟩ aufhört – in der Definition steht da nämlich ein Leerzeichen am Ende der Parameterliste.

```
    \svnInfo #1 %
```

Dann rufen wir `\ProvidesFile` aus dem LaTeX-Kernel auf.

```
    \ProvidesFile%
```

Als erster Parameter wird der Dateiname übergeben, der von `\svnInfo` ermittelt wurde.

```
    {\svnInfoFile}%
```

Dann das, wofür wir das ganze eigentlich machen: Das Datum (im Format YYYY/MM/DD, weil das von den LaTeX-Paketen auch so gemacht wird), ein Leerzeichen, dann die Versionsnummer (mit einem `v` davor). Schließlich hängen wir noch ⟨*kurzbeschreibung*⟩ an.

```
157     [\svnInfoYear/\svnInfoMonth/\svnInfoDay\space
158       v\svnInfoRevision\space #2]%
```

`\ProvidesFile` definiert damit jetzt ein Makro (`\ver@`⟨*dateiname*⟩) mit diesem Text als Inhalt, welches später von `\@dofilelist` (und unserem `\@writefilelist`) verwendet wird.

Außer dem Versionsstring wollen wir uns (falls dies per Package-Option eingeschaltet wurde) noch die Seitennummer merken, bei der die Datei anfing – das macht `\addtofilelist`, welches in Abschnitt 2.2 definiert wurde, und von `\ProvidesFile` aufgerufen wird.

```
159 }
```

2.3.2 RCS-Modus

```
160 \else%  \if@dateiliste@subversion
```

Im RCS-Modus laden wir das Paket rcsinfo.

Wir verwenden die Parameter `nofancy`, weil sonst die Fußzeile umgestellt wird, und `notoday`, weil sonst das aktuelle Datum umgestellt wird.

```
161 \RequirePackage[nofancy, notoday]{rcsinfo}
```

`\ProvideFileInfos` `{`⟨*id-string*⟩`}{`⟨*kurzbeschreibung*⟩`}`

```
162 \newcommand*{\ProvideFileInfos}[2] {%
```

Zunächst lassen wir `\rcsInfo` den ⟨*id-string*⟩ analysieren. Dies definiert (unter anderem) die Makros `\rcsInfoFile` (der Dateiname), `\rcsInfoDate` (Datum, im `YYYY/MM/DD`-Format) und `\rcsInfoRevision` (die Versionsnummer).

Das Leerzeichen nach dem `#1` ist notwendig, damit `\rcsInfo` erkennt, wo der ⟨*id-string*⟩ aufhört - in der Definition steht da nämlich ein Leerzeichen am Ende der Parameterliste.

```
163     \rcsInfo #1 %
```

Dann rufen wir `\ProvidesFile` aus dem LaTeX-Kernel auf.

```
164     \ProvidesFile%
```

Als erster Parameter wird der Dateiname übergeben, der von `\rcsInfo` ermittelt wurde. Mittels `\expandafter\@firstofone` entfernen wir dabei noch das von `\rcsInfo` (zumindest in meiner Version) eingebaute Leerzeichen am Anfang (welches ja einen anderen Namen ergibt und damit verhindern würde, dass die Information der richtigen Datei zugeschrieben wird).

```
165     {\expandafter\@firstofone\rcsInfoFile}%
```

Dann das, wofür wir das ganze eigentlich machen: Das Datum, ein Leerzeichen, dann die Versionsnummer (mit einem `v` davor). Schließlich hängen wir noch ⟨*kurzbeschreibung*⟩ an.

```
166     [\rcsInfoDate\space v\rcsInfoRevision\space #2]%
```

`\ProvidesFile` definiert jetzt ein Makro (`\ver@`⟨*dateiname*⟩) mit diesem Text als Inhalt, welches später von `\@dofilelist` (und unserem `\@writefilelist`) verwendet wird. Außer dem Versionsstring wollen wir uns (falls dies per Package-Option eingeschaltet wurde) noch die Seitennummer merken, bei der die Datei anfing.

```
}
\fi%  \if@dateiliste@subversion
```

2.4 Dateiliste

Da die Liste ziemlich lang (länger als eine Seite) werden kann, verwende ich longtable statt der eingebauten (oder der von array verbesserten) `tabular`-Umgebung. Und damit ich in der letzten Spalte nicht die Breite fest einstellen muss, sondern einfach die restliche Breite (abhängig von Seitenbreite und der Breite der anderen Spalten, welche ja abhängig vom Inhalt ist) nehmen kann, lade ich ltxtable, welches longtable mit tabularx kreuzt (und beide Pakete auch lädt).

```
\RequirePackage{ltxtable}
```

`\dateiliste@preInclude`
`\dateiliste@postInclude`

Diese beiden Macros werden vor bzw. nach dem Laden (und setzen) der Dateiliste aufgerufen. Sie sorgen dafür, dass `'` innerhalb der Liste nicht mehr in den Verbatim-Mode schaltet, wie das von pauldoc eingestellt wird. Deswegen werden sie auch nur dann so definiert, wenn pauldoc geladen wurde. (Und weil `\@ifpackageloaded` nur in der Präambel erlaubt ist, müssen wir die beiden Befehle schon zu Beginn des Dokumentes definieren, anstatt einfach die Abfrage dann zu machen, wenn es gebraucht wird.)

```
\AtBeginDocument{%
   \@ifpackageloaded{pauldoc}{%
      \newcommand*{\dateiliste@preInclude}{\DeleteShortVerb{\'}}%
      \newcommand*{\dateiliste@postInclude}{\MakeShortVerb{\'}}%
   }{%
      \newcommand*{\dateiliste@preInclude}{\relax}%
      \newcommand*{\dateiliste@postInclude}{\relax}%
   }%
}%
```

Die beiden Makros kann man sich auch selbst umdefinieren, falls andere Pakete Inkompatibilitäten ergeben.

2.4.1 Ausgabe der Liste

`\printFileList` `[`⟨*gliederung*⟩`]`

Der Vorgabewert für ⟨*gliederung*⟩ ist `\section*`, also ein unnummerierter Abschnitt.

```
\newcommand*{\printFileList}[1][\section*] {% \printFileList
```

Zunächst überprüfen wir, ob `\listfiles` in der Präambel gegeben wurde. Dies zeigt sich darin, dass das Kommando `\dofilelist` definiert ist. Andernfalls gibt es eine Warnung, und wir machen nichts.

```
\@ifundefined{@dofilelist}
{%
   \PackageWarning{dateiliste}
   {
      \protect\printFileList\space works only if
      \protect\listfiles\space is given in the preamble.
   }
}%
{%                        else (\@ifundefined{@dofilelist})
```

Andernfalls beginnen wir einen neuen Abschnitt (oder ein Kapitel oder was auch immer mit ⟨*gliederung*⟩ festgelegt wurde), mit Namen `\fileListName` und einem Label, falls man mal von wo anders darauf verweisen möchte. Danach kommt etwas beschreibender Text in `\fileListPreamble`.

```
   #1{\fileListName}\label{sec:filelist}%
   \fileListPreamble
```

In der Datei ⟨*jobname*⟩`.filelist` befindet sich nach dem ersten LaTeX-Lauf der Inhalt der Tabelle (siehe unten). Wir überprüfen zunächst, ob die Datei schon existiert.

```
   \IfFileExists{\jobname.filelist}{%
```

`\dateiliste@addtofilelist` `\@addtofilelist`

Falls ja, dann definieren wir zunächst `\@addtofilelist` um, da `\LTXtable` die Datei ⟨*jobname*⟩`.filelist` mehrfach einliest, wir aber nur einen Eintrag in der Dateiliste haben wollen. Wir verwenden nicht einfach `\@gobble`, um in dem Fall, dass durch das Setzen der Datei weitere Dateien (Schriften etc.) geladen werden, diese doch aufzunehmen. (Wir vergleichen also den Dateinamen mit dem unserer Dateinamens-Datei, und rufen im Fall der Nichtübereinstimmung das Original-`\@addtofilelist` auf.)

```
      \let \dateiliste@addtofilelist = \@addtofilelist
      \def\@addtofilelist####1{%
         \edef\dateiliste@tempa{####1}%
         \edef\dateiliste@tempb{\jobname.filelist}\relax%
         \ifx\dateiliste@tempa\dateiliste@tempb
           \relax
         \else
           \dateiliste@addtofilelist{####1}
         \fi
      }%
```

`\dateiliste@preInclude` schaltet `'` als verbatim-Char ab (und das Makro `\dateiliste@postInclude` schaltet es nachher wieder an), falls pauldoc geladen wurde (ansonsten tun sie nichts, falls nicht von jemand anders neudefiniert). Die Datei selbst wird mittels `\LTXtable` geladen.

```
      \dateiliste@preInclude
      \LTXtable{\linewidth}{\jobname.filelist}%
      \dateiliste@postInclude
```

Danach stellen wir `\@addtofilelist` wieder her und fügen unsere Dateilisten-Datei auch hinzu.

```
      \let \@addtofilelist = \dateiliste@addtofilelist
      \@addtofilelist{\jobname.filelist}%
   }
```

Falls ⟨*jobname*⟩`.filelist` nicht vorhanden war, geben wir einen Hinweistext aus, dass man LaTeX noch einmal laufen lassen soll.

```
   {%
      \AtEndDocument{\PackageWarning{dateiliste}{
         Run LaTeX again to include the File list.
      }}%
   }%
```

2.4.2 Erstellen der Liste

Jetzt noch ein paar Befehle, um die Listen-Datei zu generieren ... (Wir sind immer noch innerhalb von `\printFileList`, das alles passiert also nur, wenn dieser Befehl aufgerufen wird.)

Am Ende des Dokumentes – d.h., wenn die Dateiliste vollständig gesammelt wurde – schreiben wir sie – mit den passenden Formatierungsanweisungen – in eine Datei. (Das ganze in einer Gruppe, damit nichts kaputtgeht, und temporäre Makros nachher wieder freigegeben werden.)

```
   \AtEndDocument{%
      \begingroup
      \@writefilelist
      \endgroup
   }%
```

`\@writefilelist` Eine Variante von `\@dofilelist`, die den Inhalt – als Tabellenzeilen – in die Datei ⟨*jobname*⟩`.filelist` schreibt.

```
   \newcommand*{\@writefilelist}{%  \@writefilelist
      \newwrite\dateiliste@file
      \immediate\openout\dateiliste@file = \jobname.filelist
```

Zunächst schreiben wir eine `\ProvidesFile`-Anweisung mit dem aktuellen Datum in die `.filelist`-Datei. (Das hat den Effekt, dass diese Datei selbst auch in der Liste erscheint.)

```
      \edef\dateiliste@today{%
         \the\year/\two@digits{\the\month}/\two@digits{\the\day}}%
      \immediate\write\dateiliste@file{%
         \string\ProvidesFile{\jobname.filelist}%
         [\dateiliste@today\space --- automatically %
         generated filelist]%
      }%
```

Die eigentliche Liste wird in einer `longtable` gesetzt. Diese soll drei linksbündig gesetzte Spalten (`l`) und dann eine mit einem Absatz (`X` – mittels ltxtable aus tabularx importiert), welche den restlichen Platz ausfüllt, enthalten. Damit die letzte Spalte linksbündig (statt Blocksatz) wird, verwenden wir

`>{\raggedright\arraybackslash}` als Modifikator.[2]

```
\if@dateiliste@showpages
\def\@tempa{lrll}
\else
\def\@tempa{lll}
\fi
\immediate\write\dateiliste@file{%
   \string\LTleft=0pt%
   \string\LTright=0pt%
   \string\begin{longtable}{\@tempa>{%
         \string\raggedright\string\arraybackslash}X}%
```

Die Überschrift – aus übersetzbaren Textteilen, siehe unten, bestehend – wiederholt sich auf jeder Seite (deswegen `\endhead` anstatt `\\`).

```
      \string\textbf{\fileNameName} \expandafter&%
      \if@dateiliste@showpages
      \string\textbf{\pageName} &
      \fi
      \string\textbf{\dateName} &
      \string\textbf{\verName} &
      \string\textbf{\descriptionName}
      \string\endhead%
   }%
```

Jetzt kommt die Schleife mit den einzelnen Dateien. Das ist zum Großteil abgekupfert von `\@dofilelist` aus dem LaTeX-Kernel (`ltfiles.dtx`), welches die Liste zum Terminal ausgibt.

```
\@for\@currname:=\@filelist\do{%  \@for
```

Zunächst bestimmen wir den genauen Dateinamen – d.h. wir hängen, falls nötig, ein `.tex` an. Außerdem finden wir den zugehörigen Versions-String heraus.

```
   \filename@parse\@currname%
   \edef\dateiliste@filename{%
      \filename@base.%
      \ifx\filename@ext\relax tex\else\filename@ext\fi}%
   \expandafter\let\expandafter\dateiliste@fileversion%
   \csname ver@\dateiliste@filename\endcsname%
```

Das Makro `\dateiliste@page@`⟨*name*⟩ enthält die Seitennummer der Datei ⟨*name*⟩. Falls es nicht definiert ist, wurde sie vor dem dateiliste-Package geladen, also vor der ersten Seite. Wir merken uns das Ergebnis in `\dateiliste@filepage`.

```
   \if@dateiliste@showpages
   \@ifundefined{dateiliste@page@\@currname}
   {%
     \def\dateiliste@filepage{0}%
   }{%
     \expandafter\let\expandafter\dateiliste@filepage%
     \csname dateiliste@page@\@currname\endcsname%
   }%
```

[2] wie im LaTeX-Begleiter, zweite Auflage, Beispiel 5-3-2 vorgeschlagen.

```
            \fi
```

Jetzt schreiben wir, durch `&` getrennt, die einzelnen Felder raus. De facto merken wir uns zuerst alles mittels `\edef` in der Variable `\dateiliste@zeile`.

Zunächst der Dateiname (dabei durch den Filter `\dateiliste@escapeii` geschickt, um z.B. `_` unschädlich zu machen), ...

```
            \edef\dateiliste@zeile{%
                \expandafter\dateiliste@escapeii\expandafter{%
                  \dateiliste@filename
                }
              \space& %
```

... dann die Seitennummer (falls wir diese überhaupt anzeigen wollen)

```
              \if@dateiliste@showpages
              \dateiliste@filepage
              \space& %
              \fi
```

... dann entweder ein „—“ (falls kein Versions-String gegeben wurde), ...

```
               \ifx\dateiliste@fileversion\relax
               ---
               \else
```

... oder der Versionsstring selbst, an den ersten beiden Leerzeichen durch `&` getrennt. Dafür verfüttern wir das expandierte `\dateiliste@fileversion` (also den Versionsstring) an `\dateiliste@parse@ver`. (Für den Fall, dass da nicht genug Leerzeichen drin sind, sind am Ende noch ein paar `{}` mit Leerzeichen dazwischen – die werden ja später nicht ausgegeben.)

```
               \expandafter\dateiliste@parse@ver
               \dateiliste@fileversion{} {} {} \dateiliste@parse@ver
               \fi
```

Und jetzt noch ein `\\`, um die Tabellenzeile zu beenden.

```
               \string\\}%   \edef
```

Jetzt haben wir alles in `\dateiliste@zeile` gesammelt, und können es gebündelt ausgeben:

```
            \immediate\write\dateiliste@file{%
              \dateiliste@zeile
            }%
          }%  \@for
```

Nach der Schleife beenden wir die Tabelle und schließen dann die Datei wieder.

```
      \immediate\write\dateiliste@file{\string\end{longtable}}
         \immediate\closeout\dateiliste@file
      }%
```

`\dateiliste@parse@ver` ⟨*datum*⟩␣⟨*version*⟩␣⟨*rest*⟩`\dateiliste@parse@ver`

Dieses Makro nimmt zwei durch Leerzeichen getrennte Parameter, und gibt sie, mit zusätzlichen `&`, wieder zurück. Das letzte Stück wird noch gefiltert, damit `\bla` kein Problem bildet.

TODO: Gelegentlich fehlt die Versionsnummer. Damit dann nicht der Anfang des folgenden Textes (wie `hyperref` bei hyperref.cfg oder `package` bei pstnode.sty) erscheint, sollte noch eine Abfrage hinein, ob `##2` mit `v` oder einer Ziffer beginnt. Andernfalls sollte alles in der dritten Spalte angezeigt werden. (Leider noch nicht implementiert.)

```
      \def\dateiliste@parse@ver##1 ##2 ##3\dateiliste@parse@ver{%
        ##1 & ##2 &
        % \dateiliste@general@escape{##3}{\@backslashchar}%
        % {\string\textbackslash{}}%
        \dateiliste@escapeii{##3}%
        \relax
      }%
```

Damit ist der else-Teil und auch das ganze Makro `\printFileList` zu Ende.

```
    }%
}%
```

2.4.3 Escapen gefährlicher Makros und Zeichen

Ein Problem, welches mir durch Otto Schindler und Jens Goldhammer (unabhängig voneinander) mitgeteilt wurde: Enthält der Dateiname oder die Kurzbeschreibung nicht direkt von LaTeX verarbeitbare Zeichen (im Dateinamen sind z.B. `_`, in der Kurzbeschreibung z.B. Makronamen üblich), gibt es Probleme. (Offensichtlich muss man bei Makronamen sich etwas anstrengen, um sie da rein zu bekommen, aber es geht.)

Wir müssen also diese Zeichen vor dem Rausschreiben ersetzen.

Ein (nicht fertiggestellter) Ansatz ist in Abschnitt 2.4.4 zu finden. Hier ein anderer Ansatz, der prinzipiell zu funktionieren scheint.

`\dateiliste@escapeii` `{`⟨*text*⟩`}`

Geht ⟨*text*⟩ tokenweise durch und maskiert alles, was gefährlich werden könnte.

Zunächst übergeben wir die ganze Zeichenkette an das nächste Makro, mit einigen im echten Text wohl nicht vorkommenden Begrenzern.

```
\newcommand*{\dateiliste@escapeii}[1]{%
  \dateiliste@escape@@ii#1\@nil. \@@nil\@@
}%
```

Da ein mit `#1#2` definiertes Makro (um einzelne Token abzutrennen) Leerzeichen zwischen Makro-Parametern einfach verschluckt, müssen wir zunächst eine Wort-Trennung vornehmen – dafür dieses Makro.

`\dateiliste@escape@@ii` ⟨*wort*⟩␣⟨*rest*⟩`\@@`

Wir geben das erste ⟨*wort*⟩ an das nächste Makro (`\dateiliste@escape@ii`) weiter und arbeiten dann rekursiv auf ⟨*rest*⟩.

`#2` ist das erste Token von ⟨*rest*⟩ – ist das `\@@nil`, so haben wir das letzte Wort erreicht, denn das Leerzeichen ist jenes, das von `\dateiliste@escapeii` hinter `\@nil.` eingefügt wurde. Das ist dann also unsere Abbruchbedingung, wir müssen nur noch ⟨*wort*⟩ abarbeiten.

Ansonsten geben wir nach dem ersten Wort ein Leerzeichen aus (ein solches hatten wir ja zwischen ⟨*wort*⟩ und ⟨*rest*⟩), und rufen uns selbst mit ⟨*rest*⟩ auf.

```
\def\dateiliste@escape@@ii#1 #2#3\@@{%
  \ifx\@@nil#2
    % nur ein Wort, und das '\@nil' ist schon in '#1' drin.
    \afterfi{%
      \dateiliste@escape@ii#1\@@%
    }%
  \else
    %Rekursion
    \afterfi{%
      \dateiliste@escape@ii#1\@nil\@@%
      \space%
      \dateiliste@escape@@ii#2#3\@@%
    }%
  \fi
}%
```

`\dateiliste@escapeii` ⟨*token*⟩⟨*rest*⟩`\@@`

Maskiert ⟨*token*⟩ (per Übergabe an `\dateiliste@escapetoken`), und arbeitet danach rekursiv mit ⟨*rest*⟩.

Irgendwo in ⟨*rest*⟩ taucht ein `\@nil` auf – wenn wir bei dem angekommen sind, sind wir fertig.

Das `\@empty` ist notwendig, falls ein leerer Parameter (also `{}`) als `#1` übergeben wurde, da ansonsten `\@nil` mit `\else` verglichen wird (und der `\else`-Teil also nicht ausgeführt wird). So wird in diesem Fall `\@nil` mit `\@empty` verglichen (falsch), und der else-Teil tritt ein.

```
\def\dateiliste@escape@ii#1#2\@@{%
  \ifx#1\@nil
    \@empty
  \else
    \afterfi{%
      \dateiliste@escapetoken{#1}%
      \dateiliste@escape@ii#2\@@
    }%
  \fi
}%
```

`\dateiliste@escapetoken` `{`⟨*token*⟩`}`

Dieses Makro maskiert ein einzelnes Token für die Ausgabe in eine Datei und nachheriges Wiedereinlesen in die Tabelle.

```
\newcommand*{\dateiliste@escapetoken}[1] {%
```

Zunächst der Test, ob ⟨*token*⟩ zufälligerweise die leere Zeichenkette ist. Falls ja, vergleicht das `\ifx` nicht etwa `#1` mit `\@empty`, sondern `\@empty` mit `\@empty`, und wir machen gar nichts (weil gleich danach das `\else` kommt).

```
    \ifx#1\@empty\@empty
    \else
```

Ansonsten versuchen wir herauszufinden, ob ⟨*token*⟩ eine Kontrollsequenz o.ä. ist – dann verwendet `\ifcat` die Kategorie 16 (das ist keine der normalen Kategorien), wie auch bei `\relax`.

In diesem Fall geben wir `\string` und eine String-Darstellung der Kontrollsequenz aus, wodurch diese nach dem Einlesen angezeigt (und nicht interpretiert) wird.

```
    \ifcat\noexpand\relax
          \noexpand#1%
      \string\string\string#1%
    \else
```

Ansonsten haben wir es mit einem einzelnen Zeichen zu tun.

Wir überprüfen, welchen `\catcode` das Zeichen hätte, wenn es jetzt eingelesen würde. Hoffentlich ist das der gleiche, der auch beim Einlesen der Dateiliste eingestellt ist, und zumindest für 0-9 und 14 auch die normalen Zeichen. Wenn nicht, dann hat jemand schon gewaltig die Catcodes verbogen, und ist selbst schuld.

(Es ist leider nicht wirklich möglich, die Fallunterscheidung direkt danach zu machen, welchen Catcode das Zeichen nun hat, außer durch Vergleiche mit anderen Zeichen – und das setzt wieder vorraus, dass diese Zeichen noch ihre Original-Codes haben.)

```
    \ifcase \catcode\expandafter`#1 %
      \string\textbackslash\space% (kann nur mit Tricks vorkommen) - \
      \or\string\{% dito - {
      \or\string\}% dito - }
      \or\string\$% math shift (3) - $
      \or\string\&% alignment tab (4) - &
      \or(Kategorie 5)% end of line
      \or\string\#% parameter (6) - sollte nicht vorkommen
      \or\string\^{ }% superscript (7)
      \or\string\_% subscript (8)
      \or(Kategorie 9 - ignored)%  (9) - sollte nicht vorkommen
      \or#1% space (10)
      \or#1% letter (11)
      \or#1% other (12)
      \or\string\string\string#1% active -- sollte auch normalerweise
      %                           nicht vorkommen.
      \or\%% comment (14) (dafür muss man auch etwas tricksen)
      \or(Kategorie 15 - invalid)% (15 - sollte nicht vorkommen)
      \else(andere Kategorie)%  (und da es eigentlich nur
      %                           Kategorien 0-15 gibt, auch das nicht.)
    \fi%  (\ifcase)
    \fi%  (\ifcat)
    \fi%  (\ifx)
}%
```

Hier noch ein Hilfs-Makro, geklaut[3] aus gmutils[4]:

`\afterfi` `{`⟨*cmd*⟩`}`

Führt ⟨*cmd*⟩ nach dem nächsten `\fi` aus – alles dazwischen wird übersprungen. Damit kann man zu tiefe `\if`-Verschachtelungen im Falle einer Rekursion vermeiden.

Achtung: es geht hier wirklich um das lexikalisch nächste `\fi`, auch wenn das eigentlich zu einem dazwischen kommenden `\if` gehört. In unserem Fall kommen solche pathologischen Fälle nicht vor – falls doch, verwende man analog definierte `\afteriffifi` etc. – ich verweise auf die Dokumentation von gmutils.

```
\long\def\afterfi#1#2\fi{%
  \fi#1%
}
```

`\@@nil` `\@nil` `\@@` Damit die `\ifx`-Vergleiche bei den Escape-Funktionen funktionieren, müssen diese drei Makros unterschiedlich definiert sein. Der exakte Wert ist nicht wichtig, da sie ja nie wirklich expandiert werden (sollten).

```
\newcommand*\@@nil{\@@nil}%
\newcommand*\@nil{\@nil}%
\newcommand*\@@{\@@}%
```

2.4.4 Alternatives Escapen

```
%<*escape-original>
```

`\dateiliste@general@escape` `{`⟨*text*⟩`}{`⟨*suche*⟩`}{`⟨*ersetze*⟩`}` Dieses Makro könnte (wenn es denn funktionieren würde) von `\@writefilelist` innerhalb eines `\edef` oder `\write` aufgerufen werden, um gefährliche Zeichen zu escapen, so dass sie nach dem Einlesen im Dokument anzeigbar sind. Da es innerhalb des `\edef` sitzt, muss es expandierbar sein, darf also kein `\def` etc. enthalten. (Aha, da ist das Problem.)

Zwei Ansätze:

1. Wir versuchen, unser `\dateiliste@general@escape` expandierbar zu machen.
2. Wir bauen die API um, so dass es nicht zu seinem Ergebnis expandiert, sondern dieses in einem (expandierbaren) Makro ablegt. Entsprechend muss es dann *vor* dem `\write` aufgerufen werden, und das Ergebnis-Makro wird dann an das `\write` verfüttert.

Der erste Ansatz wurde etwas abgewandelt als `\dateiliste@escapeii` in Abschnitt 2.4.3 umgesetzt, und ist auch der, welcher jetzt verwendet wird.

[3] Ich habe absichtlich nur dieses Makro hier kopiert, anstatt das gmutils-Paket einzubinden, denn gmutils führt auch noch einige Redefinitionen in ganz anderen Bereichen durch, die ich nicht brauche bzw. haben will.

[4] von Grzegorz Murzynowski, auf CTAN unter `/macros/contrib/gmutils`.

Hier der erste Versuch, implementiert als einfache Ersetzungsfunktion. Durch den %<*escape-original>-Abschnitt wird er nicht in die Package-Datei aufgenommen. (Es wäre zusätzlich dazu auch noch eine entsprechende Änderung in \dateiliste@parse@ver sowie in \@writefilelist notwendig.)

```
\newcommand*{\dateiliste@general@escape}[3]{
  \relax
  \begingroup
  \def\dateiliste@suche{#2}%  -- dumme Fehlermeldung:
  % ! Undefined control sequence. -- warum?
  \def\dateiliste@ersetzung{#3}%
  \dateiliste@general@escape@#1\@nil\@@%
  \endgroup
}%
\def\dateiliste@gobbletoAt#1\@@{}%
\def\dateiliste@general@escape@#1#2\@@{%
  \ifx#1\@nil
  \expandafter\dateiliste@gobbletoAt
  \else
  \dateiliste@general@escape@token{#1}%
  \expandafter\dateiliste@general@escape@
  \fi
  #2\@@%
}%
\newcommand*{\dateiliste@general@escape@token}[1]{%
%  \if#3\space
%  \space%
%  \else
  \if \datailiste@suche#1%
  \dateiliste@ersetzung%
  \else
  #1%
  \fi
%  \fi
}%
```

⟨/escape-original⟩

2.4.5 Anpassbare Texte und Übersetzungen

\fileListPreamble
\fileListName
\fileNameName
\dateName
\verName
\descriptionName

Einige Namen für übersetzbare Texte – standardmäßig auf Englisch.

```
\newcommand*\fileListPreamble{
   Here is the list of all files used during the run of \LaTeX{}
   which produced this document. \footnote{More precisely, it is
      the list of files used one \LaTeX-run before the one which
      produced this document, but after some runs the list should
      stabilize.}
}
\newcommand*\fileListName{List of Files}
\newcommand*\fileNameName{file name}
\newcommand*\pageName{page}
```

```
\newcommand*\dateName{release date}
\newcommand*\verName{version}
\newcommand*\descriptionName{description}
```

`\dateiliste@babel` Hier noch gleich ein paar Übersetzungen. Wir definieren hier ein einmal-Makro, welches für mehrere Sprachen[5] zum jeweiligen Initialisierungsmakro Neudefinitionen dieser fünf Befehle hinzufügt.

```
\newcommand*{\dateiliste@babel}{
```

Zunächst Englisch - das sollte das gleiche wie die Standard-Einstellungen sein.

```
   \addto{\extrasenglish}{%
      \renewcommand*\fileListPreamble{%
         Here is the list of all files used during the run of \LaTeX{}
         which produced this document.\footnote{More precisely, it is
            the list of files used one \LaTeX-run before the one which
            produced this document, but after some runs the list
            should stabilize.}%
      }%
      \renewcommand*\fileListName{List of Files}%
      \renewcommand*\fileNameName{file name}%
      \renewcommand*\pageName{page}%
      \renewcommand*\dateName{release date}%
      \renewcommand*\verName{ver.}%
      \renewcommand*\descriptionName{description}%
   }%
```

Deutsch mit neuer Rechtschreibung.

```
   \addto{\extrasgerman}{%
      \renewcommand*\fileListPreamble{%
         Hier die Liste aller Dateien, die w\"ahrend des \LaTeX-Laufes,
         welcher dieses Dokument erzeugte, verwendet wurden.
         \footnote{genauer: Es ist die Liste aller Dokumente, die
            einen \LaTeX-Lauf fr\"uher verwendet wurden. Aber nach
            einigen L\"aufen sollte sich die Liste stabilisieren.}%
      }%
      \renewcommand*\fileListName{Liste der Dateinamen}%
      \renewcommand*\fileNameName{Dateiname}%
      \renewcommand*\pageName{\llap{Se}ite}%
      \renewcommand*\dateName{Datum}%
      \renewcommand*\verName{Ver.}%
      \renewcommand*\descriptionName{Beschreibung}%
   }%
```

Deutsch mit alter Rechtschreibung: ist das gleiche (hier tauchen keine Fälle mit Änderungen auf.)

```
   \addto{\extrasngerman}{%
      \renewcommand*\fileListPreamble{%
         Hier die Liste aller Dateien, die w\"ahrend des \LaTeX-Laufes,
```

[5] Genauer: genau für die Sprachen, welche ich soweit beherrsche, dass ich diese Texte übersetzen konnte.

```
439         welcher dieses Dokument erzeugte, verwendet wurden.
440         \footnote{genauer: Es ist die Liste aller Dokumente, die
441            einen \LaTeX-Lauf fr\"uher verwendet wurden. Aber nach
442            einigen L\"aufen sollte sich die Liste stabilisieren.}%
443      }%
444      \renewcommand*\fileListName{Liste der Dateinamen}%
445      \renewcommand*\fileNameName{Dateiname}%
446      \renewcommand*\pageName{Seite}%
447      \renewcommand*\dateName{Datum}%
448      \renewcommand*\verName{Ver.}%
449      \renewcommand*\descriptionName{Beschreibung}%
450   }%
```

Für die Verwender der Internationalen Sprache (siehe `www.esperanto.de`):

```
451   \addto{\extrasesperanto}{%
452      \renewcommand*\fileListPreamble{%
453         Jen listo de \^ciuj dosieroj, kiuj estis uzitaj dum
454         la \LaTeX-rulo, kiu produktis tiun \^ci dokumenton.
455         \footnote{Pli precize: estas la listo de dosieroj uzitaj
456            unu rulon anta\u{u} tiu, kiu produktis tiun \^ci
457            dokumenton. Sed kutime post kelkaj ruloj la listo
458            devus stabili\^gi.}%
459      }%
460      \renewcommand*\fileListName{Listo de dosieroj}%
461      \renewcommand*\fileNameName{dosiernomo}%
462      \renewcommand*\pageName{pa\^go}%
463      \renewcommand*\dateName{dato}%
464      \renewcommand*\verName{versio}%
465      \renewcommand*\descriptionName{priskribo}%
466   }%
```

Am Ende der Ausführung von `\dateiliste@babel` vernichtet der Befehl sich selbst. Das spart etwas Speicher, und sorgt dafür, dass er nicht versehentlich mehrfach ausgeführt wird (auch wenn das wohl nicht schädlich wäre).

```
467   \let \dateiliste@babel = \relax%
468 }%
```

Wir untersuchen jetzt, ob babel schon geladen wurde. Diese Fallunterscheidung ist notwendig, weil der Code von `\dateiliste@babel` zwar das Paket benötigt (also nach ihm ausgeführt werden sollte), aber nicht einfach direkt mit `\AtBeginDocument` ans Ende geschoben werden sollte, da er (falls babel schon vor diesem Paket geladen wurde) dort nach dem babel-Code (der die Sprache auswählt) kommen würde, und damit mehr nichts bewirkt.

Falls babel jetzt schon geladen wurde, ...

```
469 \@ifpackageloaded{babel}
470 {%
```

... informieren wir es sofort über die neuen Namen, die beim Sprachwechsel bitte angepasst werden sollten.

```
471   \dateiliste@babel%
```

```
}%
```

Ansonsten verschieben wir das zum Beginn des Dokumentes (und machen das auch dann nur, wenn babel inzwischen geladen wurde – ansonsten ist das ganze ja überflüssig, und `\addto` gibt es auch nicht, also können wir dann `\dateiliste@babel` vernichten).

```
{%
   \AtBeginDocument{%
      \@ifpackageloaded{babel}{%
         \dateiliste@babel%
      }{%
         \let \dateiliste@babel = \relax
      }%
   }%
}%
```

2.5 Hauptdatei in die Dateiliste

`\mainFileToList` Hier unser Nutzer-Makro für die Inklusion der Hauptdatei in die Dateiliste.

```
\newcommand*{\mainFileToList}{%  \mainFileToList
```

Falls die Dateiliste (d.h. das Makro `\@filelist`) nicht definiert ist, geben wir eine Warnung aus, und tun nichts weiter. (Ohne das würde es in dem Fall eine Endlosschleife/-rekursion geben, wenn wir versuchen, aus dieser Liste etwas zu entfernen.)

```
   \@ifundefined{@filelist}{%
      \PackageWarning{dateiliste}%
      {%
         ^^J
         \protect\mainFileToList\space (i.e. using the {dateiliste}
            package withouth^^J
         the [noaddmain] option) works only if \protect\listfiles\space
            is given^^J
         in the preamble.^^J
      }%
   }{%
```

Zunächst sehen wir nach, ob es eine Datei mit Namen ⟨*jobname*⟩`.tex` gibt.

```
   \IfFileExists{\jobname.tex} {%
      \begingroup
```

Falls ja, dann ist das höchstwahrscheinlich die Haupt-Datei des Dokumentes[6], und taucht wahrscheinlich – nämlich, wenn sie auf der Kommandozeile oder mit `\input` ⟨*jobname*⟩`.tex` anstatt `\input{`⟨*jobname*⟩`.tex}` geladen wurde – nicht in der Dateiliste auf.

[6] Andernfalls hat entweder jemand `dateiliste` manipuliert, oder die Hauptdatei ist eine Datei, die nicht auf `.tex` endet, etwa ⟨*jobname*⟩`.dtx` – in dem Fall sollte man eigentlich keine Datei ⟨*jobname*⟩`.tex` daneben haben. Ansonsten ist es wohl zumutbar, die Option `noaddmainfile` anzugeben.

Das Problem an der Erkennung des letzten Falles (LaTeX-`\input{}`) ist, dass `\jobname` die Zeichen (auch die Buchstaben) in Kategorie 12 (*other*) liefert anstatt in der natürlichen Kategorie (d.h. Buchstaben in 11 = *letter*) (wie die Dateien, deren Name irgendwo im Quelltext auftaucht und dann in `\@filelist` landet).

Ich habe drei Möglichkeiten gefunden, damit umzugehen:

(1) Wir wandeln `\@filelist` komplett in Kategorie-12-Zeichen um. [7]

(2) Wir wandeln `\jobname` (bzw. die Buchstaben darin) in ihre „richtige“ Kategorie (11 für Buchstaben) um.

(3) Wir ignorieren das Problem und leben damit, dass eventuell der Dateiname doppelt auftaucht.

Variante (2) funktioniert leider nur bei Verwendung von ε-TeX, (1) hat den Nachteil, dass anschließend `\@filelist` komplett aus Kategorie-12-Zeichen besteht, was eventuell zu Problemen mit anderen Paketen führt, welche ebenfalls diese Liste verarbeiten und auf deren Catcodes angewiesen sind.

Daher haben ich zwei Paket-Optionen hinzugefügt, welche eine entsprechende Auswahl ermöglichen:

Standardvorgehen: Falls ε-TeX verwendet wurde, nimm (2), für TeX nimm (1). (Die Unterscheidung läuft dabei natürlich nicht über die Existenz von ε-TeX, sondern über die Existenz von `\scantokens`, der beötigten Kontrollsequenz.)

`nocat12` **Mit Option `nocat12`:** Falls ε-TeX verwendet wurde, nimm (2), sonst (3).

`cat12` **Mit Option `cat12`:** Nimm immer (1).

Vor Version 0.2 gab es nur das Verhalten, welches jetzt `nocat12` entspricht.)

`\dateiliste@catxii@transform` Hier die Implementation für (1). Wir wandeln auch das `.tex` in Kategorie 12 um, mittels `\meaning` (mit `\strip@prefix` entfernen wir etwas Text, der vor der Makrodefinition steht)

```
\newcommand*{\dateiliste@catxii@transform}%
{
   \edef\dateiliste@mainfile{\jobname.tex}%
   \edef\dateiliste@mainfile{%
      \expandafter\strip@prefix\meaning\dateiliste@mainfile
   }%
```

Jetzt das gleiche für `\@filelist`.

```
   \edef\@filelist{\expandafter\strip@prefix\meaning\@filelist}%
}%
```

`\dateiliste@scantoken@tr` Hier die Implementation für (2) mit `\scantokens`: In ε-TeX gibt es dagegen den `\scantokens`-Befehl, welcher es ermöglicht, im Speicher von TeX vorliegende Token neu aus einer Pseudo-Datei einzulesen. Wenn er definiert ist (nur dann wird

[7] Den Tipp, wie das geht, habe ich zufällig beim Durchstöbern der UK-TeX-FAQ gefunden: `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=compjobnam`

dieses Makro aufgerufen), rufen wir ihn hier auf – mit einigen `\expandafter`, um nur den `\jobname` vor der `\scantokens`-Ausführung zu expandieren, und nicht die Token `\edef\dateiliste@mainfile{` und `.tex}` drumherum. `\scantokens` bekommt also die Zeichen

`\makeatletter\edef\dateiliste@mainfile{`⟨*jobname*⟩`.tex}\makeatother`

zu lesen.

Das `\edef` wird dann also mit Kategorie-11-Buchstaben (also „richtigen“) im Dateinamen ausgeführt. (Das `\makeatletter` und `\makeatother` ist notwendig, um das `@` auch als Buchstabe zuzulassen und somit `\dateiliste@mainfile` als einzelnen Makronamen anzusehen. Zu dem Zeitpunkt, zu dem `\scantokens` ausgeführt wird, sind ja die Dokumenten-Catcodes in Kraft, nicht die einer Package-Datei.)

```
\newcommand*{\dateiliste@scantoken@tr}%
{%
   \scantokens
   \expandafter{%
      \expandafter\makeatletter
      \expandafter\edef
      \expandafter\dateiliste@mainfile
      \expandafter{%
         \jobname
         .tex}%
      \makeatother
   }%
}%
```

Hier jetzt die große Fallunterscheidung:

```
\if@dateiliste@catxii
  \dateiliste@catxii@transform
\else
  \@ifundefined{scantokens}%
  {%
    \if@dateiliste@nocatxii
```

Das ist die Implementation für (3): wir merken uns den Dateinamen einfach so. Das geht wahrscheinlich schief, wenn nicht noch irgend ein anderes Makropaket eingreift, und wir können ⟨*jobname*⟩`.tex` nicht aus der Liste entfernen.

```
      \edef\dateiliste@mainfile{\jobname.tex}%
    \else
      \dateiliste@catxii@transform
    \fi
  }%
  {%
    \dateiliste@scantoken@tr
  }%
\fi
```

Jetzt haben wir in `\dateiliste@mainfile` den Namen der Hauptdatei, auch (je nach Optionen und ε-TEX-Verfügbarkeit) `@filelist` vorbereitet.

Wir können nun mit `\@removeelement` (aus dem LaTeX-Kernel) das Vorkommen von ⟨*jobname*⟩`.tex` entfernen (falls der Name dort vorhanden ist – wahrscheinlich nicht).

```
      \@expandtwoargs\@removeelement{\dateiliste@mainfile}%
      \@filelist\@filelist
```

Anschließend fügen wir den Dateinamen an den *Anfang* der Liste an.

```
      \xdef\@filelist{%
         \dateiliste@mainfile,\@filelist
      }%
      \endgroup
```

Falls ⟨*jobname*⟩`.tex` nicht existiert, ist dies sicher nicht die Hauptdatei. Dann haben wir es entweder mit einer `.dtx`-Datei zu tun (die sowieso durch das doppelte Einlesen noch einmal auftaucht), oder irgendeinen anderen Fall, den ich nicht vorhersehen kann. Also machen wir dann nichts.

```
   }{%
      \relax
   }%
  }%
}%
```

Am Ende des Dokumentes (aber noch vor dem Aufruf von `\@writefilelist`, der von `\printFileList` hinzugefügt wird) rufen wir, sofern die passende Option gesetzt war, das eben definierte Makro auf.

```
\if@dateiliste@addMain
  \AtEndDocument{\mainFileToList}
\fi
```

2.6 Schluss

```
\endinput %
⟨/package⟩
```

3 Liste der Änderungen

4 Index

Schräggedruckte Nummern verweisen auf die Seite, auf der der Eintrag beschrieben ist, unterstrichene Nummern zeigen auf die Zeilennummer der Definition, sonstige Zahlen auf die Zeilennummer einer Verwendung.

Symbols

A

C

D

E

F

I