The snapshot package

American Mathematical Society
Michael Downes

Version 2.13, 2012/08/06

1 Introduction

The snapshot package helps the owner of a LaTeX document obtain a list of the external dependencies of the document, in a form that can be embedded at the top of the document. To put it another way, it provides a snapshot of the current processing context of the document, insofar as it can be determined from inside LaTeX.

If a document contains such a dependency list, then it becomes possible to arrange that the document be processed always with the same versions of everything, in order to ensure the same output. This could be useful for someone wanting to keep a LaTeX document on hand and consistently reproduce an identical DVI file from it, on the fly; or for someone wanting to shield a document during the final stages of its production cycle from unexpected side effects of routine upgrades to the TeX system.

Normal usage of the snapshot package involves the following steps:

1. Add a `\RequirePackage` statement at the top of the document:

```
\RequirePackage{snapshot}
\documentclass{article}
  ...
```

2. Run LaTeX on the document. This will produce a dependency list in a file `\jobname.dep`. (I.e., if the document name is `vermont.ltx`, the dependency list will be named `vermont.dep`.)

3. Insert the `.dep` file at the top of the document, before `\documentclass`. The following example shows what you would end up with for a document that used the article documentclass and the graphicx package:

```
\RequirePackage{snapshot}[1999/11/03]
\RequireVersions{
  *{application}{TeX}     {1990/03/25 v3.x}
  *{format} {LaTeX2e}     {1999/06/01 v2.e}
  *{package}{snapshot}    {1999/11/03 v1.03}
  *{class}  {article}     {1999/01/07 v1.4a}
  *{file}   {size10.clo}  {1999/01/07 v1.4a}
  *{package}{graphicx}    {1999/02/16 v1.0f}
  *{package}{keyval}      {1999/03/16 v1.13}
  *{package}{graphics}    {1999/02/16 v1.0l}
  *{package}{trig}        {1999/03/16 v1.09}
  *{file}   {graphics.cfg}{0000/00/00 v0.0}
  *{file}   {dvips.def}   {1999/02/16 v3.0i}
}
\documentclass{article}
\usepackage{graphicx}
...
```

The package option `log` will cause the dependency list to appear in the LaTeX log file instead of in a separate `.dep` file:

```
\RequirePackage[log]{snapshot}
```

Making the necessary arrangements to ensure that future LaTeX runs of the document actually call in the specified versions is a separate problem. The `snapshot` package only provides a way to generate the dependency list. However, the `\RequireVersions` statement does record the given information in a form that can be accessed from within LaTeX. (It is for this purpose that it is not simply a comment.) In principle a package could be set up so that a later version would automatically attempt to emulate an earlier version if an earlier version was specified—much as LaTeX currently switches to 2.09 compatibility mode if it sees `\documentstyle` instead of `\documentclass`.

For maximum reliability font checksums should also be reported in the dependency list, but standard TeX 3.x does not provide direct access to font checksums for macro programmers. This information could be added by a separate script that scans the DVI file. (Certain nontrivial complications are possible, however.)

2 Graphics files

When a graphics file is read in by a LaTeX document using the standard `\includegraphics` command, it gets a dummy version number string of

```
Graphics file (type foo)
```

where foo is typically `eps`. This is with the current version of the `graphics` package (at the time of this writing: 1999/02/16 v1.0l). What this means in practice is that all graphics files will have their snapshot date and version number recorded as

```
Graphics v0.0
```

and will always compare equal (the string “Graphics” will be used in place of a date, but since comparison is done with `\ifx` it doesn’t make any real difference).

It would be possible, for `.eps` files at least, to read the CreationDate comment that is normally included in the file header and use that as the basis of comparison. Recording the bounding box numbers instead of a dummy version number is another possibility, which you can get with the `bbinfo` option (as of `snapshot` version 2.05).

3 Formalities

`\RequireVersions` The `\RequireVersions` command scans its argument for file names and associated version number information. The syntax of a version line for a particular file is

`*{` *file type* `}{` *file name* `}{` *version info* `}`

In other words, the `*` character in this context is like a command that takes three arguments. The extension part of the file name should be omitted in the second argument, except when the file type is `file` (following the conventions of LaTeX's `\ProvidesPackage` and `\ProvidesFile` commands). The most commonly used file types are as follows.

class A LaTeX documentclass file.

package A LaTeX package file.

tfm A TeX font metric file. In this case the "version number" is the checksum, and unless you are using an extended version of TeX this information is not accessible from inside LaTeX, so it must be filled in by an outside process. By default, font metric files are not listed in the dependency list since the checksum info is not available. There is a package option `tfm` to turn on the logging of metric files. (Not yet implemented [mjd,1999/09/23])

format This is almost always `LaTeX2e`. The information comes from `\fmtname`. Lambda, eLaTeX, and pdfLaTeX leave `\fmtname` unchanged, and although this may seem dubious at first sight, I guess they have little choice: the widespread use of `\NeedsTeXFormat` in existing package files makes them produce an error message if `\fmtname` is changed. (Maybe the thing to do would be to modify the definition of `\NeedsTeXFormat` as well.)

For a LaTeX format that uses the Babel mechanism for preloading hyphenation patterns, the version number of `babel.def` that was used in building the format might be of interest. But at first glance there does not seem to be any easy way of dealing with that, and in the normal course of things, a document that relies on Babel features will also have a `\usepackage{babel}` statement at the top, and that will yield adequate (I think) information for the snapshot dependency list.

application With standard `TeX` there is no reliable way to get the exact version number from inside LaTeX. If a document is processed with one of the recent variants that address this deficiency (such as e-TeX, pdfTeX and Omega) the available version info is used, but otherwise a presumptive value of `1990/03/25 v3.x` is used (the official release date of TeX 3.0).

file None of the above: some other file of miscellaneous type, e.g., `.clo`, `.cfg`, `.tex`, or `.def`.

The `\RequireVersions` command can be given an optional "ident" argument, similar to the argument of a `\label` command. This is not used internally but it could be used to assign a label to particular groups of files in case that helps with external processing.

4 Package options

The list of options supported by the `snapshot` package is as follows:

dep	date
log	version
tfm	major-version
warning	bbinfo
error	test
self-warning	

dep, log Write file date and version information to *jobname*`.dep` or to the LaTeX log file, respectively.

error, warning, self-warning If the `snapshot` package is invoked with the `error` option *and also* the document contains a `\RequireVersions` statement, then each subsequent `\ProvidesFile`, `\ProvidesPackage`, and `\ProvidesClass` statement will compare date and version number information with the corresponding information from the `\RequireVersions` statement and give an error message if a mismatch is detected. With the `warning` option you get warnings instead of errors. By default both the date and the version number are compared; this behavior can be modified, however, by giving additional options:

date compare only dates,
version compare only version numbers,
major-version use only the major version number when comparing.

The `self-warning` causes a warning to be given, instead of an error, if the `snapshot` package itself has a date or version mismatch.

Note: A file that doesn't have any sort of `\ProvidesFile` or `\ProvidesPackage` statement in it will show up in the dependency list, with a dummy date and version number of 0000/00/00 v0.0, but testing for a version mismatch with such a file is then infeasible.

bbinfo For files of type "graphic", include bounding-box info as the "version number". A normal date and version number are seldom available for such files, and LaTeX does not attempt to read them, which means that the `snapshot` package could not obtain the information except by drastically modifying the low-level LaTeX operations that read graphics file information—which seems overly risky.

tfm *Not implemented yet!* Include information about which TeX font metric files are called by LaTeX. This list is usually somewhat different from the list of fonts that are actually used in the output file (`.dvi` or `.pdf`), primarily because the setup for math formulas will normally preload font metric information for all the fonts from which LaTeX's basic math symbols are drawn (the symbols documented in the LaTeX book), even if the document does not use symbols from all of those fonts.

test This option is for a special purpose. It drastically changes the action of the `\RequireVersions` command so that it does not merely record the information for later reference, but does a "trial load attempt" for each file in the list, and then stops the LaTeX job as soon as the list is finished, without continuing any further.

By "trial load attempt" I mean that the `\RequireVersions` command will actually input each file, but with various bits redefined so that `\ProvidesPackage` and variants will execute `\endinput`—in other words, only the first few lines of each file will be read.

What this means, practically speaking, is that if you run such a test on a document, it gives a relatively quick check on two useful pieces of information without having to retypeset the entire document (which for some documents might be very tedious)

1. The actual location on your system from which the file will be loaded.
2. The date and version info from `\Provides...` line, if present.

Similar results could be obtained by a combination of `kpsewhich` and `grep`, but because the `snapshot test` option works through LaTeX, it is a system-independent method.

Caveat: If a file does not contain any `\ProvidesSomething` line, it will be read in its entirety, which might lead to errors. Or if there is any weird stuff preceding the `\ProvidesWhatever` line. But for well-behaved files the option seemed useful enough to be worth implementing.

5 Implementation

Standard declaration of package name and date.

```
\NeedsTeXFormat{LaTeX2e}[1994/12/01]
\ProvidesPackage{snapshot}[2012/08/06 v2.13]
```

Calling the `snapshot` package in a document causes LaTeX to list the file names and versions in the TeX log or in a `.dep` file, so that the information may be easily copied into the document file. The list so generated is nothing more than a slight adaptation of the output from LaTeX's `\listfiles` command; it puts essentially the same information into a slightly more structured form so that it will be easier to use.

For the standard mechanisms that are already built into LaTeX (e.g., the handling of the second optional argument of `\LoadClass`), the de facto "version number" is the *date* given in the optional argument of a `\ProvidesClass` or similar command. Even though most `\ProvidesWhatever` commands also give something that follows the usual form of version numbers—a string of the form `v2.3`—this is only a convention, not used internally by LaTeX, and the identification string of a random loaded file is not guaranteed to include it. The `snapshot` package copies both pieces of information if available; if the second piece is not present, a dummy number `0.0` is supplied. Similarly, files that don't include any `\ProvidesWhatever` statement will get a dummy date of `0000/00/00`;

TEX system administrators who want to ensure maximal accuracy of the snapshot information should therefore make it a practice to use `\ProvidesFile` in `.cfg` files and other local files that might have an impact on the output fidelity of their documents.

A couple of shorthand forms.

```
\let\@xp=\expandafter \let\@nx=\noexpand
```

A function to compare two strings and return FT or TT (for use with `\if`).

```
\def\str@cmp#1#2\str@cmp#3{%
  \if #1#3\else F\@car\fi \str@cmp#2\str@cmp
}
\def\string@equal#1#2{%
  \str@cmp#1\relax\str@cmp#2{\relax\@gobbletwo}\@nil TT%
}
```

`\RequireVersions` Optional argument of `\RequireVersions` allows assigning a name to a particular collection of files. This might be useful for setting a TEX inputs path.

```
\newcommand{\RequireVersions}[2][]{}%
\renewcommand{\RequireVersions}[1][]{%
  \def\snap@check{\snap@compare@versions}%
  \toks@{#1}\afterassignment\snap@storem
  \let\@let@token=
}
\def\snap@storem{%
  \ifx\@let@token\bgroup
  \else
    \PackageError{snapshot}{Expected a `\@charlb' character here}\@ehc
    \@xp\@gobblefour
  \fi
  \futurelet\@let@token\snap@branch
}
\@onlypreamble\RequireVersions
```

```
\let\snap@check\@gobble
```

```
\def\snap@finish{\toks@\bgroup}
```

```
\def\snap@branch{%
  \ifx\@let@token\egroup
    \@xp\snap@finish
  \else\ifx\@let@token *%
    \let\reserved@c\snap@store@version
  \else\ifx\@let@token\@sptoken
    \lowercase{\def\reserved@c} {\futurelet\@let@token\snap@branch}%
  \else
    \let\reserved@c\snap@store@error
  \fi\fi\fi
  \reserved@c
}
```

```
\def\snap@store@error#1{%
  \PackageError{snapshot}{Expected '*' here, not '#1'}\@ehc
}
\@onlypreamble\snap@store@error
```

```
\def\snap@store@version #1#2#3#4{%
  \@xp\snap@store@b\csname snapx@#2\endcsname{#2}{#3}{#4}%
}
\@onlypreamble\snap@store@version
```

Detection of e-LaTeX, pdfLaTeX, pdfeLaTeX, and Lambda [Omega]:

```
\ifx\OmegaVersion\@@undefined
\else
  \edef\snapshotApplication{%
    {Omega}\space\space\space
    {0000/00/00 v\OmegaVersion}%
  }%
\fi
\ifx\eTeXversion\@@undefined
\else
  \edef\snapshotApplication{%
    {eTeX}\space\space\space\space
    {0000/00/00 v\number\eTeXversion\eTeXrevision}%
  }%
\fi
\ifx\pdftexversion\@@undefined
\else
  \edef\snapshotApplication{%
    \ifx\eTeXversion\@@undefined
      {pdfTeX}\space\space
    \else
      {pdfeTeX}\space
    \fi
      {0000/00/00 v0.\number\pdftexversion\pdftexrevision}%
  }%
\fi
```

If none of the above information is available, the exact version number of TeX is not accessible from inside LaTeX. We then fall back to using a nominal date of 1990/03/25, which is when version 3.0 of `tex.web` was released by Knuth.

```
\@ifundefined{snapshotApplication}{%
  \edef\snapshotApplication{%
    {TeX}\space\space\space\space\space
    {1990/03/25 v3.x}%
  }%
}{}
```

```
\def\@fmtextension{fmt}
\def\@tfmextension{tfm}
\edef\snapx@package{.\@pkgextension}
\edef\snapx@class{.\@clsextension}
```

```
\edef\snapx@format{.\@fmtextension}
\edef\snapx@tfm{.\@tfmextension}
\long\def\snapx@ignore{}
\let\snapx@application=\snapx@ignore
\let\snapx@file=\@empty
\let\snapx@end\@@end
\expandafter\let\csname snapx@-------\endcsname\snapx@end
```

For a package named `foo.sty`, this function defines `\rqv@foo.sty` to hold the date and version information.

```
\def\snap@store@b#1#2#3#4{%
  \ifx#1\snapx@end
    \@xp\snap@finish
  \else
    \ifx#1\relax \let#1\@empty\fi
    \def\@tempa##1 ##2 ##3\@nil{##1 ##2}%
    \ifx#1\snapx@application
      \@xp\xdef\csname rqv@#3#1\endcsname{\@tempa#4 v?.? ? \relax\@nil}%
    \else
      \xdef\rqv@list{\rqv@list{#3#1}}%
      \@xp\xdef\csname rqv@#3#1\endcsname{\@tempa#4 v?.? ? \relax\@nil}%
      \snap@intest{#3}{#1}%
      \ifx#1\snapx@format \snap@check{#3.fmt}%
```

Test if current file is `snapshot.sty`. Need to pre-expand the extension part to ensure the test is correct.

```
      \else
        \edef\@tempa{\@nx\string@equal{snapshot.sty}{#3#1}}%
        \if\@tempa \snap@selfcheck \fi
      \fi
    \fi
  \fi
  \futurelet\@let@token\snap@branch
}
\@onlypreamble\snap@store@b
```

Default setup is geared to write the dependency list to a `.dep` file. The option `log` means write it to the TEX log instead.

```
\def\snap@write{\immediate\write\snap@out}
\let\snap@out\sixt@@n % fallback, probably never used
\DeclareOption{dep}{%
  \def\snap@write{\immediate\write\snap@out}%
}
\DeclareOption{log}{%
  \let\snap@write\typeout
}
```

The purpose of the 'test' option is to support a separate testing procedure for resolving file names and pre-checking version numbers. See §**??** for more information.

```
\let\snap@intest=\@gobbletwo
\DeclareOption{test}{\def\snap@intest{True}}
```

For each font used by a document, we would like to list the `.tfm` file name and checksum. If TeX provided a `\fontchecksum` primitive similar to `\fontname` that could be used to get the checksum of any font, it would just about be feasible to do this entirely from within LaTeX. As a partial solution we could at least generate the list of font file names, to make it easier for an external utility to add the checksums.

In practice, extracting font names and checksums from the `.dvi` file will probably work well enough, leaving no work to be done by the `snapshot` package in this area. But theoretically speaking the output of a document could be affected by font metric files that are loaded during LaTeX processing but that do not show up in the `.dvi` file.

```
\DeclareOption{tfm}{%
  \typeout{Option 'tfm' not implemented yet [1999/09/23]}%
}
```

Warnings and errors.

```
\def\snap@mismatch@warning#1#2#3{\PackageWarningNoLine{#1}{#2}}
\def\snap@mismatch{\snap@mismatch@warning}

\DeclareOption{error}{%
  \def\snap@mismatch{\PackageError}%
  \def\snap@selfcheck{\snap@selfcheck@a}%
  \ifx\snap@select\@empty \let\snap@select\snap@select@all \fi
}

\DeclareOption{warning}{%
  \def\snap@mismatch{\snap@mismatch@warning}%
  \def\snap@selfcheck{\snap@selfcheck@a}%
  \ifx\snap@select\@empty \let\snap@select\snap@select@all \fi
}
```

Because the exact form of the version number is not mandated by LaTeX, just take the first two "words" delimited by spaces. And take a little extra care to properly handle multiple spaces between the words.

```
\def\snap@select@all#1#2 #3#4 #5\@nil{#1#2 #3#4}
\let\snap@select\@empty
```

If the naming conventions seem a little peculiar here, it's because I had to add some pieces later that I didn't think of initially, and I wanted to minimize the chances of compatibility problems for client packages [mjd,2002-11-04].

```
\def\snap@seldate#1#2 #3\@nil{#1#2}%
\def\snap@selversion#1#2 #3{\snap@select@version #3}%
\def\snap@selmajor#1#2 #3{\snap@select@major #3}%

\DeclareOption{date}{\let\snap@select=\snap@seldate}

\def\snap@select@version#1{%
  \ifodd 0#11 \@xp\snap@sva\@xp#1\else\@xp\snap@select@version\fi
}
```

```
\def\snap@sva#1.#2 #3\@nil{#1.#2}
\def\snap@select@major#1{%
  \ifodd 0#11 \@xp\snap@svm\@xp#1\else\@xp\snap@select@major\fi
}
\def\snap@svm#1.#2\@nil{#1}
```

```
\DeclareOption{version}{\let\snap@select\snap@selversion}
\DeclareOption{major-version}{\let\snap@select\snap@selmajor}
```

```
\def\snap@bbinfo{01}
\DeclareOption{bbinfo}{\def\snap@bbinfo{00}}
```

Give this an inert definition, for the time being, until we are ready to do the split.

```
\let\snap@splitter=?
\AtBeginDocument{%
  \xdef\@filelist{\@filelist\snap@splitter}%
}
```

```
\let\snap@selfcheck\@empty
\let\snap@selfcheck@a\@empty
```

The `self-warning` option would normally be used in conjunction with the `error` option.

```
\DeclareOption{self-warning}{%
  \def\snap@selfcheck{%
    \begingroup
    \def\snap@mismatch{\snap@mismatch@warning}%
    \snap@selfcheck@a
    \endgroup
  }
}
```

```
\ExecuteOptions{warning}
\ProcessOptions\relax
```

We need the following patch to make up for the fact that `\@pkgextension` and `\@clsextension` are marked in the LaTeX kernel as "only preamble".

```
\edef\snap@restore@extensions{%
  \def\@nx\@pkgextension{\@pkgextension}%
  \def\@nx\@clsextension{\@clsextension}%
}
```

Pad filename strings out to 8+3 length so that the list will look pretty.

```
\def\snap@pad#1#2#3#4#5#6#7#8#9{\snap@pad@a{#1#2#3#4#5#6#7#8#9}}
\def\snap@pad@a#1#2#3#4#5\@nil{\snap@pad@b#1#2#3#4\space\@nil}
\def\snap@pad@b#1\space#2\@nil#3{\def#3{#2}}
```

First stage: discard leading spaces before the first and second nonspace strings in the argument. Take the first nonspace string as the date. Since we only do equal/not-equal testing on dates, it does not seem essential to test if it is really a valid date string or not (yyyy/mm/dd).

```
\def\snap@trim@version#1#2 #3{#1#2 \snap@trim@b #3}
```

Second stage: Scan for a version number. In order to handle some idiosyncratic cases, such as url.sty version 1.4, we can't simply take the second nonspace string as the version number but need to look for a leading digit.

```
\def\snap@trim@b#1{\ifodd 0#11 v#1\@xp\snap@trim@c\fi \snap@trim@b}
```

Arg 1 here is `\snap@trim@b`, which we just need to discard.

```
\def\snap@trim@c#1#2 #3\@nil{#2}
```

```
\let\rqv@list=\@empty
```

If `\fmtname.fmt` is not already in the file list, add it.

```
\edef\@tempc#1\fmtname{#1\fmtname}\@tempc
\def\@tempa#1,\fmtname.fmt,#2#3\@nil{#2}
\edef\@tempb{\@nx\@tempa,\@filelist,\fmtname.fmt,}
\if ?\@tempb?\@nil
  \edef\@filelist{\fmtname.fmt,\@filelist}%
  \def\@tempc{LaTeX2e}%
  \@xp\edef\csname ver@\fmtname.fmt\endcsname{%
    \fmtversion\space
    v\ifx\@tempc\fmtname 2.e\else ?.?\fi
  }%
\fi
```

Ensure that files get recorded.

```
\listfiles
```

```
\def\snap@doit#1{%
  \begingroup
  \ifx\delimiter#1\delimiter
  \else
    \filename@parse{#1}%
    \let\@tempd\@empty
    \ifx\filename@ext\relax
      \def\@tempa{file}\def\@tempb{~~~}%
    \else\ifx\filename@ext\@pkgextension
      \def\@tempa{package}\let\@tempb\@empty
    \else\ifx\filename@ext\@clsextension
      \def\@tempa{class}\def\@tempb{~~}%
    \else\ifx\filename@ext\@fmtextension
      \def\@tempa{format}\def\@tempb{~}%
    \else\ifx\filename@ext\@tfmextension
      \def\@tempa{tfm}\def\@tempb{~~~~}%
    \else
      \def\@tempa{file}\edef\@tempd{.\filename@ext}%
      \def\@tempb{~~~}%
    \fi\fi\fi\fi\fi
    \@xp\let\@xp\@tempe
      \csname ver@\filename@base %
        \ifx\filename@ext\relax\else.\filename@ext\fi\endcsname
    \ifx\@tempe\@empty \let\@tempe\relax \fi
    \edef\@tempe{%
```

```
      \ifx\@tempe\relax 0000/00/00 v0.0%
      \else
        \@xp\@xp\@xp\snap@trim@version\@xp\@tempe\space v0.0 v0.0 \@nil
      \fi
    }%
    \edef\@tempc{\filename@area\filename@base\@tempd}% full file name
    \@xp\snap@pad\@tempc\space~~~~~~~~~~~~~~~~\@nil\@tempd
    \let~\space
    \snap@write{\space\space *{\@tempa}\@tempb{\@tempc}\@tempd{\@tempe}}%
  \fi
  \aftergroup\snap@doit
  \endgroup
}%
```

```
\def\snap@bracify#1#2,{%
  \ifx\@empty#1\expandafter\@gobble\else {#1#2}\fi \snap@bracify
}
\def\snap@splitter@a{%
  \iffalse{{\fi }}% close current file name, end definition
  \xdef\specific@files{%
    \iffalse}\fi
    \specific@files
    \expandafter\@gobble\string % discard one closing brace
}
```

```
\def\snap@fdcheck#1{%
  \ifx\delimiter#1\@xp\@gobble
  \else\snap@fda#1\@empty.fd\@empty ?\@nil
  \fi
  \snap@fdcheck
}
\def\snap@fda#1.fd\@empty#2#3\@nil{%
  \if ?#2%
    \xdef\specific@files{\specific@files {#1}}%
  \else
    \xdef\general@files{\general@files {#1.fd}}%
  \fi
}
```

```
\let\general@files\@empty
\let\specific@files\@empty
```

The `\SpecialInput` command is related to the `packages-only` option. Apart from some ad hoc handling for `.fd` files that get loaded on demand, all files that are input after `\begin{document}` are put into the specific-files list, and all files before `\begin{document}` go into the general-files (packages-only) list. If there is a macro file for a book (say), that contains definitions specific to that book, and that is loaded in the preamble, loading it with `\SpecialInput` will cause it to go in the specific-files list.

```
\newcommand{\SpecialInput}[1]{%
  \xdef\specific@files{\specific@files{#1}}%
```

```
  \@@input#1\relax
}
```

Our definition of `\@dofilelist` does not retain much resemblance to the original in the LaTeX kernel.

```
\def\@dofilelist{%
  \snap@restore@extensions
  \xdef\general@files{\@xp\snap@bracify \@filelist \@empty,\@empty,}%
  \let\snap@splitter\snap@splitter@a
  \xdef\general@files{\general@files}%
  \let\@tempa\specific@files \global\let\specific@files\@empty
  \@xp\snap@fdcheck\@tempa{\delimiter}%
  \ifx\rqv@list\@empty
  \else \rqv@compare@lists
  \fi
  \ifx\snap@write\typeout
  \else
    \newwrite\snap@out
    \immediate\openout\snap@out=\jobname.dep \relax
  \fi
  \snap@write{\string\RequireVersions\@charlb}%
  \snap@write{\space\space *{application}%
    \snapshotApplication
  }%
  \@xp\snap@doit\general@files{\delimiter\aftergroup\@gobble\@gobble}%
  \ifx\specific@files\@empty
  \else
    \snap@specific
  \fi
  \snap@write{\@charrb}%
  \ifx\snap@write\typeout
  \else \immediate\closeout\snap@out
    \typeout{Dependency list written on \jobname.dep.}%
  \fi
}%
```

```
\def\snap@specific{%
  \snap@write{ \space *{-------}{Document-specific files:}{----}}%
  \@xp\snap@doit\specific@files{\delimiter\aftergroup\@gobble\@gobble}%
}
```

The `\rqv@compare@lists` function checks to see if any files are found only in the RequireVersions list or only in the `\general@files` list.

```
\def\rqv@condense#1{%
  \@xp\ifx\csname ver@#1\endcsname\N
  \else
    \edef\L{\L{#1}}%
    \@xp\let\csname ver@#1\endcsname=\N
  \fi
  \rqv@condense
}
```

```
\def\rqv@condend{\endcsname ?\fi
  \@xp\@xp\@xp\@gobbletwo\csname @xp\iftrue}
\def\rqv@overloaded#1{%
  \snap@mismatch{snapshot}{^^J%
    File #1 loaded though not in \noexpand\RequireVersions list%
  }\@ehc
}
\def\rqv@notloaded#1{%
  \snap@mismatch{snapshot}{^^J%
    File #1 [\csname rqv@#1\endcsname] required but not loaded%
  }\@ehc
}
\def\rqv@set#1{\@xp\let\csname ver@#1\endcsname\N \rqv@set}
\def\rqv@test#1{\csname ver@#1\endcsname{#1}\rqv@test}
\def\rqv@compare@lists{%
  \begingroup
```

Clear up duplicate file names (just in case) to avoid redundant warning messages. This should seldom be necessary in practice.

```
  \def\N{1}\let\L\@empty
  \@xp\rqv@condense\rqv@list\rqv@condend
  \global\let\rqv@list=\L
  \def\N{2}\let\L\@empty
  \@xp\rqv@condense\general@files\rqv@condend
  \global\let\general@files=\L
```

Let's make a shorthand for the code that terminates our recursion.

```
  \def\T{\@firstoftwo{\endcsname\@empty\@gobbletwo}}%
```

Set all the loaded general files to an error function.

```
  \let\N\rqv@overloaded \@xp\rqv@set\general@files \T
```

Set all the required files to an ignore function.

```
  \let\N\@gobble \@xp\rqv@set\rqv@list \T
```

Execute all the general files.

```
  \@xp\rqv@test\general@files{\endcsname\csname @gobbletwo}%
```

And now do essentially the same thing in the reverse direction.

```
  \let\N\rqv@notloaded \@xp\rqv@set\rqv@list \T
  \let\N\@gobble \@xp\rqv@set\general@files \T
  \@xp\rqv@test\rqv@list{\endcsname\csname @gobbletwo}%
  \endgroup
}
```

Compensate for a bug in old versions of `amsgen.sty`. This is a little tricky.
Old version: `\ver@amsgen`=1996/10/29 v1.2b
New version: `\ver@amsgen.sty`=1999/11/30 v2.0

```
%\@namedef{ver@amsgen.sty}{1996/10/29 v1.2b}
\AtBeginDocument{%
```

```
  \@ifundefined{ver@amsgen}{}{%
    \@xp\let\csname ver@amsgen.sty\@xp\endcsname
                    \csname ver@amsgen\endcsname
  }%
}
```

Because `\ProvidesFile` is used in `.fd` files which are normally read with special catcodes, there tend to be problems with whitespace characters being erroneously lost from the second argument. Since we have to put in a `\snap@check` call anyway, while we're at it let's fix a bug of this type that affected some older versions of LaTeX.

```
\def\ProvidesFile#1{%
  \def\snap@checker{\snap@check{#1}}%
  \begingroup
    \aftergroup\snap@checker
    \catcode`\ 10
```

Added guards from 2001/06/01 version of LaTeX. These are necessary because, for example, `inputenc` sets `\endlinechar` to a large (nonvalid character) value when reading input encoding files. The guards prevent an "invalid character" error.

```
    \ifnum\endlinechar < 256
        \ifnum \endlinechar>\m@ne
            \catcode\endlinechar 10
        \fi
    \fi
    \@makeother\/%
    \@makeother\&%
    \kernel@ifnextchar[{\snap@providesfile{#1}}{\snap@providesfile{#1}[]}%
}
```

Normally the string found in the second arg of `\ProvidesFile` (for a non-graphics file) would begin with the usual date string. The `\includegraphics` command, however, begins the second arg with `Graphic file` instead. This test therefore just checks if the first two letters are `Gr`; this is enough, ordinarily, for us to conclude that we are dealing with a graphic file.

```
\def\snap@graphic@test#1#2#3\@nil{r\if G#1#2\else X\fi}
```

```
\def\snap@providesfile#1[#2]{%
  \wlog{File: #1 #2}%
  \if\snap@graphic@test#2@@\@nil
    \snap@record@graphic#1\relax #2 (type ??)\@nil
  \else
    \expandafter\xdef\csname ver@#1\endcsname{#2}%
  \fi
  \endgroup
}
```

This is what `\includegraphics` does to record graphic file information.

```
\@providesfile #1[#2]->
\wlog {File: #1 #2}\expandafter \xdef \csname ver@#1\endcsname {#2}
```

```
\endgroup
#1<-\Gin@base \Gin@ext
#2<-Graphic file (type eps)
```

Check the graphics info.

```
\def\snap@record@graphic#1\relax #2(type #3)#4\@nil{%
  \expandafter\xdef\csname ver@#1\endcsname{%
    Graphic%
    \if\snap@bbinfo :bb=\Gin@llx/\Gin@lly/\Gin@urx/\Gin@ury\fi
    \space v0.0%
  }%
}
```

```
\def\@pr@videpackage [#1]{%
  \expandafter\xdef\csname ver@\@currname.\@currext\endcsname{#1}%
  \ifx\@currext\@clsextension
    \typeout{Document Class: \@gtempa\space#1}%
  \else
    \wlog{Package: \@gtempa\space#1}%
  \fi
  \snap@check{\@currname.\@currext}%
}
```

```
\def\snap@selfcheck@a{\snap@check{snapshot.sty}}
```

```
\def\@nofmt#1.fmt.#2 {#1 }
```

```
\def\snap@mismatch@a#1#2#3{%
  \snap@mismatch{snapshot}{^^J%
    \space\space Required version #2 of \@nofmt#1.fmt. and^^J%
    \space\space provided version #3 do not match%
  }\@ehc
}
```

When comparing `\rqv@foo.sty` (information from a previous LaTeX run) with `\ver@foo.sty` (information from current run), we first call `\snap@trim@version` on the latter to clear away any idiosyncrasies in the contents.

```
\def\snap@compare@versions#1{%
  \begingroup
    \@ifundefined{rqv@#1}{}{%
      \edef\0{\csname rqv@#1\endcsname}%
      \edef\1{\csname ver@#1\endcsname}%
      \edef\1{\@xp\snap@trim@version\1 v0.0 v0.0 \@nil}%
      \edef\@tempa{\@xp\snap@select\0 v0.0 v0.0 \@nil}%
      \edef\@tempb{\@xp\snap@select\1 v0.0 v0.0 \@nil}%
      \ifx\@tempa\@tempb
      \else
        \edef\@tempd{\@nx\snap@mismatch@a{#1}{\@tempa}{\@tempb}}%
        \@xp\@tempd
      \fi
    }%
  \endgroup
```

When the `test` option is in effect, jump out of the current file instead of continuing.

```
  \snap@test@abort
}
```

```
\let\snap@test@abort=\@empty
```

6 Compatibility

Suppose that I have a LaTeX document containing a `\RequireVersions` statement generated by `snapshot` and I send this to my colleague who, we believe, has a LaTeX setup that is for our purposes identical. Suppose that our belief is erroneous in the following way: My colleague has a newer version of `snapshot` and a newer version of one of the affected files.

Here is what we **don't** want to happen: That the differing version of snapshot would cause the other differing file to be accepted without demur.

Conversely, if it is I who have the newer version of `snapshot`, the main concern is that some difference in the contents of the `\RequireVersions` statement would lead to an error when my colleague attempts to process the document.

7 In conclusion

```
\ifx\snap@select\@empty
  \let\snap@compare@versions\@gobble
  \let\snap@check\@gobble
\fi
```

Fallback for a command that is sometimes used in AMS journal production.

```
\providecommand{\controldates}[1]{}
```

8 And finally . . .

If the embedded `\RequireVersions` data in a LaTeX document is extracted to a separate file, and

```
\RequirePackage[test]{snapshot}
```

is added at the top, then the file can be run as a small separate LaTeX job that will, among other things, produce in the log file a nice list of fully resolved file names—sort of a limited, but system-independent variant of the `kpsewhich` idea.

```
\ifx\snap@intest\@gobbletwo \endinput \fi
```

Some old, ill-behaved packages might throw in a `\makeatother` at the end which can cause problems for the next file that comes along when testing.

```
\def\restore@some@catcodes{}
\def\save@some@catcodes{%
  \edef\restore@some@catcodes{%
    \catcode\number`\@=\number\catcode`\@
    \catcode\number`\"=\number\catcode`\"
    \catcode\number`\^=\number\catcode`\^
    \catcode\number`\_=\number\catcode`\_
    \relax
  }%
}
```

Some typical calls of `\snap@intest`:

```
\snap@intest{LaTeX2e}{\snapx@format}
\snap@intest{snapshot}{\snapx@package}
\snap@intest{mcom-l}{\snapx@class}
\snap@intest{amsmath}{\snapx@package}
\snap@intest{umsa.fd}{\snapx@file}
\snap@intest{pictex}{\snapx@file}
```

The extant public versions of the following files (in the teTeX distribution, at least) are known to be problematic when we are trying to read a `\ProvidesWhatever` line from the top of the file: psfig.sty, pictex.sty, pictex.tex, epic.sty, amstex.sty, xy.tex. Either (a) they don't have a `\ProvidesWhatever` line at all, or (b) they include some code before the `\ProvidesWhatever` line that makes some assumption true in normal processing but false in snapshot-test processing. E.g., there is code in `amstex.sty` that assumes `\documentstyle` or `\documentclass` was already executed and that the `\if@compatibility` switch got set accordingly.

```
\def\snap@intest#1#2{%
  \message{^^J}%
  \begingroup \edef\0{#1#2}\def\9{latex209.def}%
  \ifx\0\9\global\@compatibilitytrue \fi
  \ifx#2\snapx@format
```

If arg1 + arg2 = "LaTeX2e.fmt", the calling function `\snap@storeb` will run `\snap@check` separately. This is a crude way of making things work in that case without much extra trouble.

```
    \def\snap@test@abort{\endgroup}%
  \else
    \edef\N{%
      \noexpand\snap@intest@b{#1#2}%
        {#1}{\@xp\@gobble#2\@empty}%
        {\csname rqv@#1#2\endcsname}}%
    \expandafter\endgroup\N
  \fi
}
```

```
\def\snap@intest@b#1#2#3#4{%
  \def\@currname{#2}\def\@currext{#3}%
  \begingroup \lccode`\/=`\0\relax\lowercase{\endgroup
  \ifnum\snap@seldate#4 00 0\@nil>\z@
    }% matches \lowercase
    \save@some@catcodes
    \@@input #1 \relax
    \restore@some@catcodes
  \else
    \snap@specialtest{#1}{#4}%
  \fi
}
\def\snap@specialtest#1#2{%
  \fake@input{#1}%
}
\def\fake@input#1{%
  \begingroup
  % Ensure that outer \foo or unmatched braces don’t trip us up
  \catcode‘\\=12 \catcode‘\{=12 \catcode‘\}=12
  \endinput
```

Note that these definitions of `\G` and `\?` are local, and recall that one-letter cs names don’t use up hash table entries.

```
  \def\G{\@car\endgroup}%
  \expandafter\futurelet\expandafter\?\expandafter\G\@@input#1 \relax\@nil
}
\let\snap@test@abort=\endinput
\let\snap@selfcheck=\@empty
```

There’s an extra close-brace left hanging around at the end, but I guess we don’t care.

```
\def\snap@finish{%%
  \endgroup \message{^^J}%
  \def\X##1{##1,\X}%
  \edef\@filelist{\@xp\X\rqv@list{\@gobbletwo}}%
  \def\X##1,?{##1}\edef\@filelist{\@xp\X\@filelist ?}%
  \@dofilelist
  \@@end
}%
\def\snap@mismatch#1#2#3{}
```

Problematic: xy.sty, because it calls xy.tex before it calls `\ProvidesPackage`. And pictex.tex because it doesnt use `\ProvidesFile` at all.

```
\renewcommand{\RequireVersions}[2][]{%
  \begingroup
  \makeatletter
  \def\snap@check{\snap@compare@versions}%%
  \let\snapx@tfm=\snap@ignore
```

This seems to help, with english.ldf for example, to prevent an endless loop when attempting to load babel.def.

```
  \def\ProvidesLanguage##1{\ProvidesFile{##1.ldf}}%
  \iffalse{\fi \futurelet\@let@token\snap@branch #2}%
  \endgroup
}
```

The usual `\endinput` to ensure that random garbage at the end of the file doesn’t get copied by `docstrip`.

```
\endinput
```