The filecontentsdef package

Jean-François Burnol
jfbu (at) free (dot) fr
Package version: v1.4 (2019/04/20); documentation date: 2019/04/21.
From source file filecontentsdef.dtx (21-04-2019 at 15:24:14 CEST)

Abstract

This lightweight LaTeX2e package provides an environment `filecontentsdef` which is like the `filecontents` environment of Scott Pakin's filecontents package but in addition to the file creation stores the (verbatim) contents into a macro given as an additional argument (either as a control sequence or as a name).

Displaying verbatim these contents is possible via `\filecontentsprint`, and executing them (if they represent LaTeX code) via `\filecontentsexec`.

A variant environment `filecontentsdefmacro` stores the contents into a macro, but skips the save-to-a-file part.

I developed this to display TEX code verbatim in documentation and simultaneously produce during the LaTeX run the corresponding files in order to embed them in the PDF as *file attachment annotations* (via the services of Scott Pakin's further package attachfile.)

1 Description

filecontentsdef

This package provides the `filecontentsdef` environment. It is like `filecontents` but requires a second argument. This argument will either be:

- a single control sequence token such as `\macro`,

New with v1.4→

- or anything else which then must after expansion be usable as a macro *name* (it will be handled via `\csname...\endcsname` encapsulation). For example `\myitemnumber{3}` can be used as argument and it will then be expanded inside `\csname...\endcsname` to construct a control sequence, whose name will possibly contain digits or other characters of non-letter catcodes. A single active character is allowed as long as its expansion is `\csname...\endcsname` compatible; the character itself will *not* be assigned a new meaning.

Thus the syntax is either:

```
\begin{filecontentsdef}{<filename>}{\macro}
... arbitrary contents ...
\end{filecontentsdef}
```

or:

```
\begin{filecontentsdef}{<filename>}{(expanding to) macro name}
... arbitrary contents ...
\end{filecontentsdef}
```

The environment creates the file and stores its (verbatim) contents into its second argument `\macro` (or into the macro `\<macro name>` with given name).

1. The starred variant acts like `filecontents*` environment regarding the file contents.
2. The contents put into the macro are the same for the starred and non-starred environments: they do not contain the commented-out header.
3. The scope of the macro definition is global.
4. No check is done on whether the defined macro pre-existed.
5. The macro holds a verbatim rendering of the contents, which handles the Form Feed and Tabulation characters analogously to the `filecontents` treatment.
6. Babel shorthands will be neutralized the same way they are when encountered in a verbatim environment or in a `filecontents` environment. Their action is reactivated if the macro (assuming the contents represent LaTeX code) gets later executed via `\filecontentsexec`.
7. The environment can be used either in the preamble or the body of the document.
8. The contents must not contain themselves a `\end{filecontentsdef}` (or `end{filecontentsdef*}` in the starred case).
9. If some `\macro` contains the *name* of the macro to be defined, use then `\empty\macro` as argument, this trick will avoid the environment thinking it is supposed to redefine `\macro` itself.

Here are some additional relevant details:

1. the usual special characters are sanitized like they would be in a verbatim environment,
2. the space becomes the active character of ascii code `32`,
3. the end of line is converted into the active character `^^M` (i.e. ascii code `13`),
4. the tabulation character `CTRL-I` becomes the active space character,
5. the form feed character `CTRL-L` is converted into a pair of active `^^M`,
6. the active bytes of ascii code between `128` and `255` (this is now systematically the case with `inputenc+utf8` being default) are stored into the produced macro "as is",
7. the non-active bytes of ascii code between `128` and `255` are stored into the produced macro with catcode letter.

These last two items together mean that 8bit or UTF8-encoded characters will display as expected in a verbatim rendering, see `\filecontentsprint` next.

`\filecontents-print`

The `\filecontentsprint` macro has a unique mandatory argument which will be either:

- a single control sequence token (for example `\macro`),

New with v1.4→

- or anything else which then must after expansion be usable as a macro *name* (for example `macro`). It will be handled via `\csname...\endcsname` encapsulation (see earlier explanations).

The \macro must be of the type constructed by the environment filecontentsdef. It will be then be printed exactly as

```
\begin{verbatim}
<contents of \macro>
\end{verbatim}
```

would have done.

This uses underneath the verbatim environment and has been tested to be compatible with the standard verbatim, with the one from package doc (classes ltxdoc.cls, scrdoc.cls) and also with the one from package verbatim (whose mechanism is quite different from the one of the default verbatim environment.)

> Due to limitation of the verbatim environment, the \end{verbatim} must not appear inside the contents... else it will be misconstrued as ending the external verbatim environment itself which is added by \filecontentsprint!
>
> This limitation only affects \filecontentsprint, not \filecontentsexec.

\filecontentsexec

Although filecontentsdef itself generally does not require ε-TEX, it provides as a convenience \filecontentsexec which does require it as it uses \scantokens to re-assign the current catcode regime to the verbatimized tokens stored into its mandatory argument. Again the mandatory argument may have one of the two forms described previously. New with v1.4→

And of course this assumes that the tokens provide legitimate LATEX code.

TeX-hacker note: As an aftereffect, the \newlinechar will be set to 10 (its LATEX default) after execution; we skipped the saving and restoring of its value (it is temporarily set to 13 for \scantokens execution). No group is used in order to not create an extra scoping of the executed macro contents.

TeX-hacker note: A space token will be present at the end of the expansion, due to \scantokens's way of working, but may be hidden by the nature of the contents themselves, if for example they end with a \relax or a %, or from arising in vertical mode.

filecontentshere

The environment

```
\begin{filecontentshere}{<filename>}
... arbitrary contents ...
\end{filecontentshere}
```

creates on the fly a file with these contents, and simultaneously it typesets them in a verbatim environment. It is a shortcut to doing

```
\begin{filecontentsdef}{<filename>}{\filecontentsheremacro}
... arbitrary contents ...
\end{filecontentsdef}
```

and then immediately

```
\filecontentsprint\filecontentsheremacro
```

The \filecontentsheremacro remains available for reuse with \filecontentsexec.

For example

```
\begin{filecontentshere}{\jobname.test}
  \begin{framed}
    \noindent
                We have coded this in \LaTeX: both
      $E=mc^2$ (input as \verb|$E=mc^2$|)
    and     $E=h\nu$  owe much    to \textsc{Albert Einstein}.
  \end{framed}
\end{filecontentshere}
\filecontentsexec\filecontentsheremacro
```

will produce the external file with extension `.test` and have this effect in the document (verbatim framed then real framed):

```
\begin{framed}
  \noindent
              We have coded this in \LaTeX: both
    $E=mc^2$ (input as \verb|$E=mc^2$|)
  and     $E=h\nu$  owe much    to \textsc{Albert Einstein}.
\end{framed}
```

> We have coded this in LaTeX: both $E = mc^2$ (input as `$E=mc^2$`) and $E = h\nu$ owe much to Albert Einstein.

filecontents-defmacro

The environment

```
\begin{filecontentsdefmacro}{\macro or macro name}
... arbitrary contents ...
\end{filecontentsdefmacro}
```

New with v1.4

was added at `v1.4`. It is like `filecontentsdef` without the "save to file" part... and has thus a sole mandatory argument which may be either a control sequence or a name (or material expanding to name), as previously described. The macro can then be either printed verbatim via `\filecontentsprint` or, if it consists of LaTeX code, be executed in re-tokenized form via `\filecontentsexec`.

Its name is thus a bit paradoxical but was chosen to share an existing prefix with the other package macros and environments.

2 How to wrap usage of filecontentsdef in another environment

It is simple: don't use `\begin/\end` syntax but `\begingroup\filecontentsdef{..}{..}` and `\endfilecontentsdef\endgroup`. And these should come last, respectively first, in the definition of the begin, respectively end, part of the new environment.

The extra `\begingroup...\endgroup` is needed only to the extent that definitions and catcode assignments could affect the execution of the rest of the `end` part of the new environment. As this `end` part will be tokenized at time of definition, it will in

general be indifferent to the catcode modifications done by \filecontentsdef and generally speaking the \begingroup...\endgroup can be dropped.

Of course, for wrapping the starred variant one should use \csname filecontentsdef*\endcsname, or make the definition with * having catcode letter. But the \endfilecontentsdef can drop the *, as the starred environments defined by the filecontentsdef package use the same ending macros as their non-starred variants.

As exercise, let's imagine we want an environment which will be associated to some counter, will automatically increment it at each usage, and will use this counter to index the files and macros created on each invocation.

```
\newcounter{pablo}
\newenvironment{defexercise}
   {\stepcounter{pablo}%
    \begingroup
    \csname filecontentsdef*\endcsname
    {\jobname-ex\the\value{pablo}}{exercise-\the\value{pablo}}}%
   {\endfilecontentsdef\endgroup}
\newcommand{\printexercise}[1]{\filecontentsexec{exercise-\the\numexpr#1\relax}}
```

We can then use it this way:

```
\begin{defexercise}
   Prove that \[x^n+y^n=z^n\]  is not solvable in positive integers if $n$ is at
   most $-3$.\par
\end{defexercise}
\begin{defexercise}
   Refute the existence of black holes in less than 140 characters.\par
\end{defexercise}
\begin{defexercise}
  \def\NSA{NSA}%
  Prove that factorization is easily done via probabilistic algorithms and
  advance evidence from knowledge of the names of its employees in the
  seventies that the \NSA\ has known that for 40 years.\par
\end{defexercise}
\begin{itemize}
\item \printexercise{3}
\item \printexercise{2}
\item \printexercise{1}
\end{itemize}
```

This produces in the document:

- Prove that factorization is easily done via probabilistic algorithms and advance evidence from knowledge of the names of its employees in the seventies that the NSA has known that for 40 years.
- Refute the existence of black holes in less than 140 characters.
- Prove that
$$x^n + y^n = z^n$$

is not solvable in positive integers if n is at most -3.

Additionally, three small files were created containing the LaTeX mark-up for each exercise.

3 Custom verbatim print-outs

We now comment on alternatives to using `\filecontentsprint`. Here is some (non-LaTeX) text snippet.

```
\begin{filecontentsdef}{\jobname.test2}{\testactive}
v1.2 \[2016/09/19\]
-------------------

Initial version.

test: éèàùÉÈÇÀÙÛÎåðñòóôõöœøùúûüýþßŸŽ§
\end{filecontentsdef}
```

We can expand `\testactive` directly inside the LaTeX document, but must give some definitions to the active space token and the active `^^M` token like `verbatim` environment does.

For a true verbatim printout `\obeyspaces` and `\obeylines` are not enough because spaces at start of lines will disappear, and multiple empty lines give multiple `\par`'s which collapse into a single one (hence no empty line can be observed in the output). The usual `verbatim` environment uses a special definition of `\par` which prevents the disappearance of empty lines, and for the spaces it has macro `\@vobeyspaces` which makes the spaces issue `\leavevmode` so they are not skipped at the start of lines. Let's define:

```
\makeatletter
% this redefines active spaces, but does not make spaces active
\def\niceactivespaces{\@vobeyspaces\catcode32=10\relax}%
\makeatother
\begingroup
% this redefines active end of lines, but does not make them active
  \catcode`\^^M\active %
  \gdef\niceactiveCRs{\def^^M{\leavevmode\par}}%
\endgroup %
```

Then we can issue something like:

```
{\setlength{\parindent}{2cm}\niceactivespaces\niceactiveCRs\testactive\par}
```

This allows hyphenation and ligatures, which are usually inhibited in standard `verbatim`, and it does not switch to the monospace font. Here is what happens if we do all of the above, as a test:

v1.2 \[2016/09/19\]
———————-

Initial version.

test: éèàùÉÈÇÀÙÛÎåðñòóôõöœøùúûüýþßŸŽ§

We see indeed how the `---...--` gave rise to ligatures, and that the monospace font was not used. This was only to give an idea of how one can use the macros created by the `filecontentsdef` or `filecontentsdefmacro` environments for variant verbatim rendering. This technique can be used as workaround to the problem with `\filecontentsprint` that the contents can not contain `\end{verbatim}`.

4 Implementation

See the `README.md` file for the `CHANGE LOG`.

```
\NeedsTeXFormat{LaTeX2e}[1999/12/01]
\ProvidesPackage{filecontentsdef}
 [2019/04/20 v1.4 filecontents + macro + verbatim (JFB)]
```

Most of the code is still identical to the one in Scott Pakin's filecontents hence to the original one in LaTeX's sources.

But `v1.4` adds `\filecontentsdefmacro` which does not write to a file...

```
\begingroup
\catcode`\^^M\active%
\catcode`\^^L\active\let^^L\relax%
\catcode`\^^I\active%
\gdef\filec@ntentsdef#1#2{%
  \let#2\@empty%
 \if@filesw%
  \openin\@inputcheck#1 %
  \ifeof\@inputcheck%
    \@latex@warning@no@line%
        {Writing file `\@currdir#1'}%
  \else%
    \@latex@warning@no@line%
        {Overwriting file `\@currdir#1'}%
  \fi%
  \closein\@inputcheck%
  \chardef\reserved@c15 %
  \ch@ck7\reserved@c\write%
  \immediate\openout\reserved@c#1\relax%
  \if@tempswa%
    \immediate\write\reserved@c{%
      \@percentchar\@percentchar\space%
          \expandafter\@gobble\string\LaTeX2e file `#1'^^J%
      \@percentchar\@percentchar\space  generated by the %
        `\@currenvir' \expandafter\@gobblefour\string\newenvironment^^J%
      \@percentchar\@percentchar\space from source `\jobname' on %
         \number\year/\two@digits\month/\two@digits\day.^^J%
      \@percentchar\@percentchar}%
  \fi%
 \fi%
  \let\do\@makeother\dospecials%
```

SP's `filecontents` sets here in the loop all catcodes to `11`, but we need for correct rendering in verbatim that the constructed macro stores active characters as active characters.

We don't check for unusual active characters of ascii code `<128` as this is not done by original or SP's `filecontents`. But if present then they will expand similarly both in the `\write` and in the construction of the macro.

```
  \count@=128\relax%
  \loop%
    \ifnum\catcode\count@=\active%
          \lccode`~\count@%
          \lowercase{\def~{\noexpand~}}%
    \else%
          \catcode\count@=11 %
    \fi%
    \advance\count@ by \@ne%
    \ifnum\count@<\@cclvi%
```

```
\repeat%
```

The default active `^^L` is `\outer`. But `\reserved@b` will be `def'd` with an active `^^L` in its replacement text.

```
\let^^L\relax%
\edef\E{\@backslashchar end\string{\@currenvir\string}}%
\edef\reserved@b{\def\noexpand\reserved@b####1\E####2\E####3\relax}%
```

`\filecontentsdefmacro` sets `\if@filesw` to false.

```
\reserved@b{%
  \ifx\relax##3\relax%
    \if@filesw\immediate\write\reserved@c{##1}\fi%
```

This is where the original `filecontents` is extended to store the parsed material in a macro (in my very first hack I simply patched it to redefine `\write` to also do the macro storage, but considerations like the one relative to active characters due to `inputenc` made me decide to re-write the whole thing, hence make a new package.)

Active characters were defined with a single `\noexpand` in the loop, and this is enough because after each new line is processed the characters it contains are protected from further expansion in the `\xdef`'s. And the single `\noexpand` is enough also for the `\write` done above.

The `lccode` of the tilde is `32` when this gets executed. Multiple form feeds produce the same effect in the macro (insertion of two `^^M` per form feed) as in the written out file (via two `^^J`).

```
    \toks@\expandafter{#2}%
    {\def^^L{\noexpand^^M\noexpand^^M}\lowercase{\let^^I~}%
     \xdef#2{\the\toks@##1\noexpand^^M}}%
  \else%
    \edef^^M{\noexpand\end{\@currenvir}}%
    \ifx\relax##1\relax%
    \else%
     \if@filesw%
        \@latex@warning{Writing text `##1' before %
           \string\end{\@currenvir}\MessageBreak as last line of #1}%
      \immediate\write\reserved@c{##1}%
     \fi%
```

Same added code as above.

```
      \toks@\expandafter{#2}%
      {\def^^L{\noexpand^^M\noexpand^^M}\lowercase{\let^^I~}%
       \xdef#2{\the\toks@##1\noexpand^^M}}%
    \fi%
    \ifx\relax##2\relax%
    \else%
       \@latex@warning{%
         Ignoring text `##2' after \string\end{\@currenvir}}%
    \fi%
  \fi%
  ^^M}%
\catcode`\^^L\active%
```

`v1.4`: sync `^^L` and `^^I` with 2018/04/01 LaTeX release.

```
\let\L\@undefined%
\def^^L{\expandafter\ifx\csname L\endcsname\relax\fi^^J^^J}%
\catcode`\^^I\active%
\let\I\@undefined%
\def^^I{\expandafter\ifx\csname I\endcsname\relax\fi\space}%
\catcode`\^^M\active%
\edef^^M##1^^M{\noexpand\reserved@b##1\E\E\relax}%
```

We want space characters to be active in the produced macro. We only need to protect them once from expansion.

```
  \catcode32\active\lccode`~32 \lowercase{\def~{\noexpand~}}%
}%
\endgroup
```

The `v1.4` macros accept a name as alternative to a macro. Empty or ill-formed `#2` will break code. But `#2` can be using `\if`, `\else`, `\fi` tokens, the whole thing will end up `\csname`-expanded. An active character also will end up `\csname`-expanded. There is no check on whether `#2` or `\csname#2\endcsname` is an existing macro.

```
\long\def\filecontentsdef@aux#1\filecontentsdef@aux{\@firstofone}%
\def\filecontentsdef@get#1#2%
{%
  \def\@tempa{#1}\def\@tempb{{#2}}%
  \expandafter\filecontentsdef@aux\@gobbletwo#2\filecontentsdef@aux
  \@thirdofthree
  \filecontentsdef@aux
  {\ifcat\relax\noexpand#2\expandafter\@gobble\else\expandafter\@firstofone\fi}%
  {\edef\@tempb{{\expandafter\noexpand\csname#2\endcsname}}}%
  \expandafter\@tempa\@tempb
}%
```

`v1.4` adds `\filecontentsdefmacro`. I abuse the `\if@filesw` toggle as there is no `\if@tempswb` available. No need for a starred version as anyhow the commented-out header was not put into the macro by the existing `\filec@ntentsdef` code. No need to reset the toggle as only usage as an environment is supported. And by lazyness and to spare extra coding, `\endfilecontentsdefmacro` emits no warning about Form Feed and Tabulation.

```
\begingroup
\catcode`\*=11
\gdef\filecontentsdef #1{\@tempswatrue\filecontentsdef@get{\filec@ntentsdef{#1}}}%
\gdef\filecontentsdef*#1{\@tempswafalse\filecontentsdef@get{\filec@ntentsdef{#1}}}%
\global\let\endfilecontentsdef \endfilecontents
\global\let\endfilecontentsdef*\endfilecontents
\gdef\filecontentsdefmacro{\@fileswfalse\filecontentsdef@get{\filec@ntentsdef{}}}%
\global\let\endfilecontentsdefmacro\relax
\gdef\filecontentshere #1{\@tempswatrue\filec@ntentsdef{#1}\filecontentsheremacro}%
\gdef\filecontentshere*#1{\@tempswafalse\filec@ntentsdef{#1}\filecontentsheremacro}%
\gdef\endfilecontentshere{\endfilecontentsdef\aftergroup\filecontents@here}%
\global\let\endfilecontentshere*\endfilecontentshere
```

Package `verbatim.sty` modifies the standard `verbatim` environment. For both the original and the modified version we need to insert an active `^^M` upfront, else an empty first line would not be obeyed. The `verbatim.sty`'s `verbatim` needs that we feed it with the macro expanded once, as it uses active end of lines as delimiters and they thus need to be immediately visible. It also needs an active `^^M` after the `\end{verbatim}`. To avoid to check at `\AtBeginDocument` if package `verbatim.sty` is loaded, we use a slightly tricky common definition. The advantage is that this may help make the code compatible with further packages (I have not looked for them) modifying the `verbatim` environment. For better code readibility I use `^^M%`'s rather than exploiting the active ends of lines here.

```
\gdef\filecontentsprint{\filecontentsdef@get\filecontents@print}%
\catcode`\^^M\active%
\gdef\filecontents@print #1{\let\filecontents@print@EOL^^M\let^^M\relax%
   \begingroup\toks@\expandafter{#1}\edef\x{\endgroup%
            \noexpand\begin{verbatim}^^M%
            \the\toks@\@backslashchar end\string{verbatim\string}}\x^^M%
   \filecontents@print@resetEOL}%
\gdef\filecontents@print@resetEOL{\let^^M\filecontents@print@EOL}%
```

```
\endgroup
\def\filecontents@here{\filecontents@print\filecontentsheremacro}%
\def\filecontentsexec{\filecontentsdef@get\filecontents@exec}%
\def\filecontents@exec #1{\newlinechar13
    \scantokens\expandafter{#1}\newlinechar10\relax}%
\endinput
```