The filecontentsdef package

Jean-François Burnol
jfbu (at) free (dot) fr

Package version: v1.5 (2019/09/29); documentation date: 2019/09/29.
From source file filecontentsdef.dtx (29-09-2019 at 10:46:12 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.)

Contents

1 Environments and macros 1
- 1.1 filecontentsdef, filecontentsgdef . . . 1
- 1.2 filecontentsdefmacro, filecontentsgdefmacro . . . 3
- 1.3 \filecontentsprint, \FCDprintenvname, \FCDprintenvoptions . . . 3
- 1.4 \filecontentsprintviascan . . . 5
- 1.5 \filecontentsexec . . . 6
- 1.6 filecontentshere . . . 7

2 How to wrap usage of filecontentsdef in another environment 8

3 How to customize handling of tabulation and form feed characters 9

4 How to obtain customized verbatim in the output 11

5 Related packages 12

6 Implementation 13

1 Environments and macros

1.1 filecontentsdef, filecontentsgdef

The `filecontentsdef` environment is like `filecontents` but requires a second argument. This argument will either be:

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

- 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>}{\foo}
... 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 `\foo` (or into the macro `\<macro name>` with given name).

> Hint: if some `\foo` contains the *name* of the macro to be defined, use `\empty\foo` as argument, thus avoiding `\foo` itself to get overwritten by the environment.

1. The starred variant `filecontentsdef*` acts like `filecontents*` environment regarding the file contents, i.e. it drops addition of a (TEX) commented out header.

New with v1.5

2. `filecontentsdefstarred` is an alias for `filecontentsdef*`.
3. The contents put into the macro are the same for the starred and non-starred environments: neither contains a commented-out header.

Changed at v1.5

4. The scope of the macro definition is **local**.

New with v1.5

5. Use `filecontentsgdef` for a definition with global scope.
6. No check is done on whether the defined macro pre-existed.
7. The macro holds a verbatim rendering of the contents, with active spaces and active `^^M` tokens.

New with v1.5

8. The handling of the Form Feed and Tabulation characters can be, contrarily to the `filecontents` treatment, customized. See section 3.
9. 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`.

10. The environment can be used either in the preamble or the body of the document.

11. The contents must not contain themselves a `\end{filecontentsdef}` (or `\end{filecontentsdef*}` in the starred case).

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. see section 3 for the handling of the horizontal tabulation and form feed characters,
5. 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",
6. 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.

1.2 filecontentsdefmacro, filecontentsgdefmacro

This environment

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

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.

1.3 \filecontentsprint, \FCDprintenvname, \FCDprintenvoptions

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

- a single control sequence token (for example `\foo`),
- 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 `\foo` must be of the type constructed by the environments `filecontentsdef` or `filecontentsdefmacro`. It will be then be printed exactly as

```
\begin{verbatim}
<contents of \foo>
\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`.

New with v1.5 The name of the used environment is held in macro `\FCDprintenvname`. Redefine it to modify the environment name from its default `verbatim`.

New with v1.5 Furthermore the macro `\FCDprintenvoptions` can be used to pass options to that environment.

Here is a set-up using `fancyvrb` which I tested with success:

```
\documentclass{article}
\usepackage{filecontentsdef}
\usepackage{xcolor}
\usepackage{fancyvrb}
\usepackage{fvextra}

% store tabs as active characters so they can be handled by fancyvrb
{\catcode`\^^I=\active\gdef\FCDtabtomacro{\noexpand^^I}}

% Use fancyvrb Verbatim environment
\renewcommand*\FCDprintenvname{Verbatim}

% with these options. This will be submitted to an \edef, so we
% simply wrap in \unexpanded to avoid problems
\renewcommand*\FCDprintenvoptions{%
  \unexpanded{[fontsize=\scriptsize, highlightlines={1, 3-4},
               numbers=both, showspaces, spacecolor=red,
               showtabs, %tab=\rightarrowfill% incompatible with linebreaks?
               breaklines,breakbefore=\\\space]}%
}
\begin{filecontentsdefmacro}{\testfancyvrb}
some contents with long lines and tabs to test the options
\end{filecontentsdefmacro}
```

Here is now a set-up with `minted` which worked also (although the syntax highlighting was not handling correctly control sequences using the @ letter, but I am not knowledgeable enough in the Pygmentize library):

```
\documentclass{article}% needs shell-escape
\usepackage{filecontentsdef}
```

```
\usepackage{minted}
\newminted{tex}{linenos}

\renewcommand*\FCDprintenvname{texcode}

\begin{filecontentsdefmacro}{\testminted}
some TeX macros
\end{filecontentsdefmacro}

\begin{document}
\filecontentsprint\testminted
\end{document}
```

1.4 \filecontentsprintviascan

It is not possible to use a `listings` environment via `\filecontentsprint` and `\FCDprintenv`, because `listings` has a special way to identify where it ends and this is incompatible with the `\filecontentsprint` approach.

New with v1.5

At `v1.5` `\filecontentsprintviascan` is added which is more powerful than `\filecontentsprint` as it should work not only with verbatim environments known to be compatible with `\filecontentsprint` but also with environments of the `listings` type. But this requires ε-TEX `\scantokens` (it uses `\filecontentsexec`, see next).[1]

Here is an example of usage which worked for me (although the syntax highlighting, like the one from `minted`, was not fully satisfying):

```
\documentclass{article}
\usepackage{filecontentsdef}
\usepackage{xcolor}
\usepackage{listings}
\lstnewenvironment{latexverbatim}
 {\lstset{
     basicstyle=\small\ttfamily,
     breaklines=true,
     columns=fullflexible,
     language=[LaTeX]TeX,
     numbers=left,
     numbersep=1em,
     numberstyle=\tiny\color{gray},
     keywordstyle=\color{red}}}{}

\renewcommand*\FCDprintenvname{latexverbatim}

\begin{document}
```

[1] I know that the LaTeX kernel itself requires ε-TEX for a few years now, but `filecontentsdef` could have been installed on an old machine...

```
\begin{filecontentsdefmacro}{\testlistings}
\gdef\filecontentsprintviascan{\FCD@get\FCD@printviascan}%
\gdef\FCD@printviascan#1{%
   \toks@\expandafter{#1}%
   \edef\FCD@envwithcontents{%
        \noexpand\begin{\FCDprintenvname}\FCDprintenvoptions\noexpand^^M%
        \the\toks@\@backslashchar end{\FCDprintenvname}\noexpand^^M}%
   \FCD@exec\FCD@envwithcontents}%
\end{filecontentsdefmacro}

\filecontentsprintviascan\testlistings
\end{document}
```

1.5 \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.

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

TeX-hacker note: No group is used in order to not create an extra scoping of the executed macro contents.

TeX-hacker note: In the case of storage in a macro of some LaTeX contents, and re-parsing via `\filecontentsexec` which uses `\scantokens`, the last line before the `\end{filecontentsdef}` or `\end{filecontentsdefmacro}` potentially generates an end of line character, typically a space token (but this depends on the `\endlinechar` valid at this location setting). Such a space coming from the last end of line has no impact if the LaTeX contents get executed in vertical mode. In horizontal mode it will be avoided if these contents end with a `%` or also with some control sequence such as a `\relax` or `\empty` (only if `\endlinechar` is standard).

New with v1.5

At `v1.5` a special convention is added that if the `\end{filecontentsdef}` (or variants) stands on the same line as the last line of the contents, rather than being on a line of its own, then the stored contents will get postfixed with an added `\empty` token. Thus execution via `\filecontentsexec` in horizontal mode will not induce any ending space token (assuming the `\endlinechar` is then at its standard setting). This feature is to be used only with contents representing TeX macros, as the `\empty` makes sense only in that context. This token will remain invisible in the PDF output from `\filecontentsprint`.

Changed at v1.5

No `\empty` gets added to the last line of the exported file (if there is one). The original `filecontents` environment issues a warning when there is contents before the end of the environment on the same line, this warning is now skipped.

This extra `\empty` added only under such special usage of the environment is not currently customizable, but I can add such a feature if there is a user request.

TeX-hacker note: one should not think that using `\filecontentsexec` with some stored material will behave like copying pasting that material directly at that very location of the source code: for example if the last line ends with a `%` this does not mean that this will comment out what is next in the source code on the same line after the `\filecontentsexec`! It is more analogous to making first a macro definition with the contents and then execute that macro. But it is not possible for some macro from inside these contents to itself grab tokens coming next after the `\filecontentsexec\foo` (moreover, **filecontentsdef** adds tokens of its own in order restore the `\newlinechar`).

1.6 filecontentshere

This 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` is then available for usage as argument of `\filecontentsexec`.

Changed at v1.5 Since `v1.5` its definition has only local scope.

The environment has a starred variant `filecontentshere*` (also `filecontentsherestarred`) which does the expected thing.

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 an external file with the above contents 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.

2 How to wrap usage of filecontentsdef in another environment

Don't use the \begin/\end syntax but directly \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 are mandatory with the **filecontentsdef** environments making a local scope definition.

Changed at v1.5

For those creating global scope macros, \begingroup...\endgroup is recommended, as the LaTeX state is not completely clean after \endfilecontentsgdef (et al.) execution. Thus it is better to have the extra \begingroup...\endgroup pair always (which are a part of the things added by the \begin/\end syntax).

For wrapping the starred variant one needs to use \csname filecontentsdef*\endcsname, or make the definition with * having catcode letter. A simpler way is to use the alias ending in ...starred. Regarding the ending macro its name can drop the *, as the starred environments defined by the **filecontentsdef** package use the same ending macros as their non-starred variants.

New with v1.5

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. Except if you know how to smuggle how a macro from an environment you probably want to use filecontentsgdef in order for the macro to have global scope.

```
\newcounter{pablo}
\newenvironment{defexercise}
   {\stepcounter{pablo}%
    \begingroup
    \filecontentsgdefstarred
    {\jobname-ex\the\value{pablo}}{exercise-\the\value{pablo}}}%
   {\endfilecontentsgdefstarred\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 How to customize handling of tabulation and form feed characters

LaTeX assigns catcode 10 by default to `^^I` meaning that it is handled by default as a space character:

```
\def\test{^^I}
\ifx\test\space \textcolor{blue}{OK}\else \ERROR\fi
```

OK

But **filecontentsdef** like the original `filecontents` environment assigns active catcode to the tabulation character before parsing the contents. This allows special treatment.

> Attention that if input as `^^I` (in opposition to the real ascii character of ascii code 9), it will end up simply as `^^I` in external file or macro, because the caret loses its special meaning in the environment.
>
> The discussion in this section about customizing **filecontentsdef** behaviour applies only to a source with a real tabulation character, not one in TeX notation `^^I`.

With `v1.5`, **filecontentsdef** diverges from original `filecontents` by adding the means to customize the handling of such tabulation character, rather than simply raising a warning and exporting it as a space like original `filecontents`. And also the handling of the form feed character can be customized.

New with v1.5

This is controlled via four control sequences whose default definitions are the following:

```
\def\FCDtabtofile{ }%

{\catcode32\active\gdef\FCDtabtomacro{\noexpand }}%

\def\FCDformfeedtofile{^^J^^J}%

{\catcode`\^^M\active\gdef\FCDformfeedtomacro{\noexpand^^M\noexpand^^M}}%
```

Each of `\FCDtabtofile`, `\FCDtabtomacro`, `\FCDformfeedtofile`, `\FCDformfeedtomacro` gets used via an expansion inside an `\edef`, hence the need for `\noexpand` in front of active characters. The `^^J` in `\FCDformfeedtofile` matches the default newline character (ascii code 10) of LaTeX for exporting files. The active `^^M` is used for macro storage because this is the most suitable for verbatim printing via `\filecontentsprint` as typically in a verbatim environment end of lines get converted into such active `^^M` (which will create a `\par` token).

If you want for example tabulation characters to get converted into four spaces, use:

```
\def\FCDtabtofile{\space\space\space\space}%
{\catcode32\active\gdef\FCDtabtomacro{\noexpand \noexpand \noexpand \noexpand }}%
```

I have here used for macro storage active spaces. The `\noexpand` are mandatory in such case, but they will disappear from stored contents in a macro. We could also have defined `\FCDtabtofile` as an alias to `\FCDtabtomacro` here because a non-expanding active space will simply give a space in the external file. Due to TeX tokenization rules `\def\FCDtabtofile{ }` would be the same as with only one single space and thus we used `\space\space\space\space`.

If you want an actual tabulation character stored to a file, use:

```
{\catcode`\^^I=12 \gdef\FCDtabtofile{^^I}}
\def\FCDtabtomacro{TAB}
```

The second line is for demonstration only as an example of how to store the tabulation character in a macro, use anything adequate to replace `TAB` as used here. For example you can store in it the actual tabulation (ascii code 9) with catcode 12, via using rather `\let\FCDtabtomacro\FCDtabtofile` in the above. Or use some Unicode symbol and appropriate font configuration.

Here is an example. This is rendered via `verbatim` which will treat the tabulation characters as spaces, but they are there in the original:

```
{\catcode`\^^I=12 \gdef\FCDtabtofile{^^I}}
\let\FCDtabtomacro\FCDtabtofile
\begin{filecontentshere}{\jobname-tab.test}
Here is a tab and then three in a row .
\end{filecontentshere}
(we will see them in the PDF output via the glyph at slot 9 of the T1 encoding).
```

Here is what the original produces indeed when executed:

```
Here is a tab¯and then three in a row¯¯¯.
```

The way it looks in the PDF is due to our definition of `\FCDtabtomacro` which gives a catcode12 character of ascii code 9, and we use T1 font-encoding. The exported file does contain on the other hand as promised a real tabulation character.

> LaTeX since 2018 uses by default `\usepackage[utf8]{inputenc}` and almost all control characters are given active catcode. Exceptions: `^^@` (illegal), `^^I` (treated as space), `^^J` (for some reason it gets catcode 12 and thus will if printed to PDF give the glyph at slot 10 of the font encoding), and `^^M` (end of line).
>
> Notice that such a control character which in the source gets input using TeX notation, for example `^^P`, causes no issue to **filecontentsdef** as the caret `^` has lost its special catcode. Thus a `^^P` will be printed to external file.
>
> If one tries to use directly in the source the CTRL-P, an error will be raised by LaTeX triggered by the active character (**filecontentsdef** does not sanitize catcodes in this ascii range, as original `filecontents` environment does not either).
>
> One can always reassign catcodes, thus you can set the catcode of `^^P` to 12 for example. However, when exporting such a control character with catcode 11 or 12 to a file, pdflatex uses `^^` notation in the output. There are three exceptions (with `pdflatex`): the horizontal tabulation (ascii code 9, `^^I`), the line feed (ascii code 10, `^^J`), and vertical tabulation (ascii code 11, `^^K`) are exported to the file as the corresponding ascii characters (at least this is the case with TeXLive). *These exceptions do not apply with* `xelatex`.
>
> Thus we again end up with TeX notation `^^P` in the exported file. To get a literal CTRL-P, you need to set the catcode of `^^P` to 11 or 12, and to run `pdflatex` with the `-8bit` option. With XeLaTeX, this would be needed even for the horizontal tabulation CTRL-I.

4 How to obtain customized verbatim in the output

Please refer first to the discussion of `\filecontentsprint` and `\filecontentsprintviascan` as it provides examples using `fancyvrb`, `minted`, or `listings` environments which may be what you are looking for.

Here we make quick comments on alternatives to using `\filecontentsprint`, and handling directly the contents via the configuration of active spaces and active end of lines.

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}`.

5 Related packages

- Scott Pakin's filecontents. Notice that the package functionality has been integrated into LaTeX release dated `2019/10/01`.
- Pablo González's scontents. Make sure your version is at least `v1.3` as earlier ones had a dependency on **`filecontentsdef`** and are broken by the changes coming with `filecontentsdef v1.5`.

6 Implementation

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

```
\NeedsTeXFormat{LaTeX2e}[1999/12/01]
\ProvidesPackage{filecontentsdef}
 [2019/09/29 v1.5 filecontents + macro + verbatim (JFB)]
\let\FCD@global\global
\begingroup
\catcode`\^^M\active%
```

Attention that all end of lines must now get protected due to the end of line character being active.

The LaTeX default for active `^^L` was `\outer` up to the `2017-01-01` release (got modified in `{v2.3b}{2016/11/06}` version of `ltplain.dtx`). But we must still handle that possibility for usage with older formats.

```
\catcode`\^^L\active\let^^L\relax%
\catcode`\^^I\active%
```

`\FCD@main` is the core construct.

Bulk of the code is still identical to the one in Scott Pakin's filecontents hence to the original one in LaTeX's sources regarding `\filec@ntents`, as `filecontentsdef` was conceived as an extension of the original `filecontents` environment, with the added feature of storing the verbatimized contents in a TeX macro, which can then be printed (verbatim) or re-tokenized later via `\scantokens` and executed (if it represents LaTeX material). Starting with `v1.5` some renaming of internal macros appears and the original coding is not only extended but starts being modified at some places as well.

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

`v1.5` renames all internal macros to use `\FCD@` namespace prefix. This applies here to `\FCD@main` whose name was still `\filec@ntentsdef` at `v1.4`. The change also applies to the `\reserved@b` and `\reservec@c`.

`v1.5` adds customizability of how tabulation and form feed characters are handled either in file output or in macro storage.

`v1.5` modifies `filecontentsdef` to only make a definition with local scope. For this we must smuggle out of the environment both the name and the meaning of the control sequence to define. We prepare for this a `\FCD@defmacro` which gets executed via `\endfilecontentsdef` (and variants). As `\FCD@defmacro` issues `\endgroup`, direct usage of `\filecontentsdef` (not as environment) must be inside an explicit `\begingroup`/`\endgroup` pair, mimicking the one which an environment would insert.

The `filecontentsgdef/filecontentsgdefmacro` environments do define `\FCD@defmacro` but do not execute it.

```
\gdef\FCD@main#1#2{%
  \def\FCD@defmacro%
     {\toks@\expandafter{#2}%
      \edef\x{\endgroup\def\noexpand#2{\the\toks@}%
              \begingroup\def\noexpand\@currenvir{\@currenvir}}%
      \x}%
  \FCD@global\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\FCD@reserved@c15 %
  \ch@ck7\FCD@reserved@c\write%
  \immediate\openout\FCD@reserved@c#1\relax%
  \if@tempswa%
    \immediate\write\FCD@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` was `\outer` up to `2017/01/01`.

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

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

```
  \FCD@reserved@b{%
    \ifx\relax##3\relax%
      \if@filesw%
        \let^^L\FCDformfeedtofile%
        \let^^I\FCDtabtofile%
        \immediate\write\FCD@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.

```
      \toks@\expandafter{#2}%
```

```
      \let^^L\FCDformfeedtomacro%
      \let^^I\FCDtabtomacro%
      \FCD@global\edef#2{\the\toks@##1\noexpand^^M}%
    \else%
      \edef^^M{\noexpand\end{\@currenvir}}%
      \ifx\relax##1\relax%
      \else%
        \if@filesw%
```

`v1.5` suppresses the warning issued by `filecontents` environment in such case.

```
          \let^^L\FCDformfeedtofile%
          \let^^I\FCDtabtofile%
          \immediate\write\FCD@reserved@c{##1}%
        \fi%
```

In such case with the end of environment not being on a line of its own, `v1.5` injects an extra `\empty` token at end of last line. Thus usage in this form is restricted to contents representing TEX macros. We still need the final `^^M` for `\filecontentsprint` matters.

```
        \toks@\expandafter{#2}%
        \let^^L\FCDformfeedtomacro%
        \let^^I\FCDtabtomacro%
        \FCD@global\edef#2{\the\toks@##1\noexpand\empty\noexpand^^M}%
      \fi%
      \ifx\relax##2\relax%
      \else%
         \@latex@warning{%
           Ignoring text `##2' after \string\end{\@currenvir}}%
      \fi%
    \fi%
    ^^M}%
```

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

`v1.5`: drop the `^^L` and `^^I` LATEX definitions in favour of `\FCDformfeedtomacro` etc...

```
  \catcode`\^^L\active%
  \catcode`\^^I\active%
  \catcode`\^^M\active%
  \edef^^M##1^^M{\noexpand\FCD@reserved@b##1\FCD@E\FCD@E\relax}%
```

We need active space characters to be active in the produced macro. We only need to protect them once from expansion. The definition will work both for writing to a file and for storage into the macro.

```
  \catcode32\active\lccode`~32 \lowercase{\def~{\noexpand~}}%
}%
\gdef\FCDformfeedtofile{^^J^^J}%
\gdef\FCDformfeedtomacro{\noexpand^^M\noexpand^^M}%
\gdef\FCDtabtofile{ }%
\catcode32\active\gdef\FCDtabtomacro{\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.

`v1.5` replaces `\@tempa,b` by `\FCD@tempa,b`. And its `\FCD@get@aux` directly grabs `#2` rather than issuing `\@firstofone`.

```
\long\def\FCD@get@aux#1\FCD@get@aux#2{#2}%
\def\FCD@get#1#2%
{%
```

```
  \def\FCD@tempa{#1}\def\FCD@tempb{{#2}}%
  \expandafter\FCD@get@aux\@gobbletwo#2\FCD@get@aux
  \@thirdofthree
  \FCD@get@aux
  {\ifcat\relax\noexpand#2\expandafter\@gobble\else\expandafter\@firstofone\fi}%
  {\edef\FCD@tempb{{\expandafter\noexpand\csname#2\endcsname}}}%
  \expandafter\FCD@tempa\FCD@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 `\FCD@main` code. No need to reset the toggle as documentation explains direct usage of the environment begin and end macros must be inside a `\begingroup\endgroup` pair.

v1.5 adds the "starred"-named variants as aliases.

```
\begingroup
\catcode`\*=11
\gdef\filecontentsdef #1{\let\FCD@global\@empty
                         \@tempswatrue\FCD@get{\FCD@main{#1}}}%
\gdef\filecontentsdef*#1{\let\FCD@global\@empty
                         \@tempswafalse\FCD@get{\FCD@main{#1}}}%
\global\let\filecontentsdefstarred\filecontentsdef*
\gdef\filecontentsgdef #1{\@tempswatrue\FCD@get{\FCD@main{#1}}}%
\gdef\filecontentsgdef*#1{\@tempswafalse\FCD@get{\FCD@main{#1}}}%
\global\let\filecontentsgdefstarred\filecontentsgdef*
```

v1.5 can not use original `\endfilecontents` which makes reference to `\reserved@c` whereas **filecontentsdef** now uses in its place `\FCD@reserved@c`. Finally I drop altogether the bulk of this macro which issued a warning in case of an encountered form feed or tabulation character.

v1.5 adds support for definitions with local scope so we must smuggle the defined macro out of the environment. This is done via `\FCD@defmacro` which is set-up via `\FCD@main`.

```
\gdef\endfilecontentsdef{\immediate\closeout\FCD@reserved@c\FCD@defmacro}%
\global\let\endfilecontentsdef*\endfilecontentsdef
\global\let\endfilecontentsdefstarred\endfilecontentsdef
\gdef\endfilecontentsgdef{\immediate\closeout\FCD@reserved@c}%
\global\let\endfilecontentsgdef*\endfilecontentsgdef
\global\let\endfilecontentsgdefstarred\endfilecontentsgdef
\gdef\filecontentsdefmacro{\let\FCD@global\@empty
                           \@fileswfalse\FCD@get{\FCD@main{}}}%
\gdef\endfilecontentsdefmacro{\FCD@defmacro}%
\gdef\filecontentsgdefmacro{\@fileswfalse\FCD@get{\FCD@main{}}}%
\global\let\endfilecontentsgdefmacro\relax
\gdef\filecontentshere #1{\let\FCD@global\@empty
                          \@tempswatrue\FCD@main{#1}\filecontentsheremacro}%
\gdef\filecontentshere*#1{\let\FCD@global\@empty
                          \@tempswafalse\FCD@main{#1}\filecontentsheremacro}%
\global\let\filecontentsherestarred\filecontentshere*
\gdef\endfilecontentshere{\endfilecontentsdef\aftergroup\FCD@here}%
\global\let\endfilecontentshere*\endfilecontentshere
\global\let\endfilecontentsherestarred\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.

`v1.5` adds `\FCDprintenvname` which holds the name of the environment to be used. Initially configured to hold of course `verbatim`. And it also adds `\FCDprintenvoptions`.

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

`v1.5` adds `\filecontentsprintviascan` which uses `\scantokens` to wrap stored contents in a verbatim-like environment. This is needed for things such as `listings` (it uses an active backslash at start of lines). Attention to the `\scantokens` subtlety with `\end` which would become `\end<space>` and then again `listings` would not recognize the ending pattern.

```
\gdef\FCD@printviascan#1{%
   \toks@\expandafter{#1}%
   \edef\FCD@envwithcontents{%
        \noexpand\begin{\FCDprintenvname}\FCDprintenvoptions\noexpand^^M%
        \the\toks@\@backslashchar end{\FCDprintenvname}\noexpand^^M}%
   \FCD@exec\FCD@envwithcontents}%
\endgroup
\def\FCDprintenvname{verbatim}%
\let\FCDprintenvoptions\@empty
\def\FCD@here{\FCD@print\filecontentsheremacro}%
\def\filecontentsexec{\FCD@get\FCD@exec}%
```

`v1.5` restores `\newlinechar` to its prior value, rather than setting it to the LaTeX default (10).

```
\def\FCD@exec #1{\edef\FCD@newlinechar{\the\newlinechar}%
    \newlinechar13
    \scantokens\expandafter{#1}\newlinechar\FCD@newlinechar\relax}%
\endinput
```