The morewrites package: Always room for a new \write*

Bruno Le Floch

2011/06/19

Contents

*This file has version number 2478, last revised 2011/06/19.

1 morewrites documentation

This LaTeX package is meant to be a solution for the error "no room for a new `\write`", which occurs when too many macro packages reserve streams to write data to various auxiliary files. It is in principle possible to rewrite packages so that they are less greedy on resources, but that is often unpractical for the end-user. Instead, morewrites hooks at the lowest level (TeX primitives). If I did my job correctly, users simply need to add the line `\usepackage{morewrites}` somewhere near the beginning of the LaTeX file, and the "no room for a new `\write`" error should be gone.

As this is a rather new package, it has not been tested very thoroughly yet. I thus encourage you to check that references are correct after loading that package: if they are correct without morewrites, but wrong with, please send me a minimal file showing the problem, or post a question on the tex.stackexchange.com question and answers website, or the comp.text.tex newsgroup.

2 Known deficiencies

Some distributions of TeX allow a quoted syntax for file names with spaces. I haven't yet coded that. A temporary fix is to avoid file names with spaces.

3 morewrites implementation

<*package>

```
\ProvidesExplPackage
  {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
\RequirePackage{expl3}
```

Let us recall the syntax of the commands that we need to modify (from Chapter 24 of the TeXbook): each of those can be preceeded by `\immediate`, which we will thus also have to modify.

- `\openout` ⟨*4-bit number*⟩ ⟨*equals*⟩ ⟨*filename*⟩
- `\write` ⟨*number*⟩ ⟨*general text*⟩ with a special case when ⟨*number*⟩ is 18.
- `\closeout` ⟨*4-bit number*⟩

Finally, we also need to modify `\newwrite` to let it allocate more than 16 streams.

First save the primitives (including those for input, because this package might be extended to cover that as well).

```
\mw_tex_openin:w
\mw_tex_read:w
\mw_etex_readline:w
\mw_tex_closein:w
\mw_tex_immediate:w
\mw_tex_openout:w
\mw_tex_write:w
\mw_tex_closeout:w
```

```
\cs_new_eq:NN \mw_tex_openin:w    \tex_openin:D
\cs_new_eq:NN \mw_tex_read:w      \tex_read:D
\cs_new_eq:NN \mw_etex_readline:w \etex_readline:D
\cs_new_eq:NN \mw_tex_closein:w   \tex_closein:D
\cs_new_eq:NN \mw_tex_immediate:w \tex_immediate:D
\cs_new_eq:NN \mw_tex_openout:w   \tex_openout:D
\cs_new_eq:NN \mw_tex_write:w     \tex_write:D
\cs_new_eq:NN \mw_tex_closeout:w  \tex_closeout:D
\cs_new_eq:NN \mw_tex_shipout:w   \tex_shipout:D
```

(*End definition for* `\mw_tex_openin:w` *and others.*)

3.1 Variables

`\g_mw_late_write_int` The integer `\g_mw_late_write_int` labels the various non-immediate operations in the order in which they appear in the source.

```
\int_new:N \g_mw_late_write_int
```

`\g_mw_writes_prop` The property list `\g_mw_writes_prop` associates a file name to each open stream.

```
\prop_new:N \g_mw_writes_prop
```

`\g_mw_read` `\c_mw_tmp_file_tl` The expansion that `\write` performs is impossible to emulate with anything else than `\write`. We will need to write to a file and read back from it for things to work properly.

```
\newread \g_mw_read
\tl_const:Nn \c_mw_tmp_file_tl { \jobname.mw }
\newwrite \g_mw_write
```

`\g_mw_reserved_writes_seq` Some of the writing streams are already allocated when loading this package, and we let LaTeX manage them.

```
\seq_new:N \g_mw_reserved_writes_seq
\prg_stepwise_inline:nnnn {0} {1} { \g_mw_write - 1 }
  { \seq_gput_right:Nn \g_mw_reserved_writes_seq {#1} }
\seq_gput_right:Nn \g_mw_reserved_writes_seq {18}
```

`\l_mw_tmpa_tl` `\g_mw_step_int` `\g_mw_tmpa_int` `\g_mw_box`

```
\tl_new:N  \l_mw_tmpa_tl
\int_new:N \g_mw_step_int
\int_new:N \g_mw_tmpa_int
\newbox \g_mw_box
```

`\g_mw_stream_int` `\g_mw_filename_tl` `\g_mw_text_tl` What parsing finds is stored in those three variables.

```
\int_new:N \g_mw_stream_int
\tl_new:N  \g_mw_filename_tl
\tl_new:N  \g_mw_text_tl
```

\mw_scan_stop: A recognizable version of \scan_stop:.

```
\cs_new_eq:NN \mw_scan_stop: \scan_stop:
```

(*End definition for* \mw_scan_stop:.)

3.2 Parsing

\mw_parse_number:Nw Stores a number in \g_mw_stream_int before performing #1.

```
\cs_new_protected_nopar:Npn \mw_parse_number:Nw #1
  {
    \tex_afterassignment:D #1
    \tex_global:D \g_mw_stream_int
  }
```

(*End definition for* \mw_parse_number:Nw.)

\mw_parse_text:Nw This is used for the \write command: to grab a ⟨*general text*⟩ we assign a \toks, within a group to avoid clobbering \toks0.

```
\cs_new_protected_nopar:Npn \mw_parse_text:Nw #1
  {
    \group_begin:
      \tex_aftergroup:D #1
      \tex_afterassignment:D \mw_parse_text_aux:
      \tex_toks:D \c_zero =
  }
\cs_new_protected_nopar:Npn \mw_parse_text_aux:
  {
      \tl_gset:Nx \g_mw_text_tl { \tex_the:D \tex_toks:D \c_zero }
    \group_end:
  }
```

(*End definition for* \mw_parse_text:Nw.)

\mw_parse_equals_filename:Nw Remove a potential = sign.

```
\cs_new_protected_nopar:Npn \mw_parse_equals_filename:Nw #1
  {
    \peek_meaning_remove:NTF =
      { \mw_parse_filename:Nw #1 } { \mw_parse_filename:Nw #1 }
  }
```

(*End definition for* \mw_parse_equals_filename:Nw.)

\mw_parse_filename:Nw Empty the filename, and build it one character at a time. We need to expand tokens as in an f-type expansion, but without losing spaces. For that, we use \peek_gafter:Nw to find

out whether the coming token is expandable or not, and distinguish further depending on what kind of non-expandable token it is.

```
\cs_new_protected_nopar:Npn \mw_parse_filename:Nw #1
  {
    \group_begin:
    \group_align_safe_begin:
    \tex_aftergroup:D #1
    \tl_gclear:N \g_mw_filename_tl
    \mw_parse_filename_loop:w
  }
\cs_new_protected_nopar:Npn \mw_parse_filename_loop:w
  { \peek_gafter:Nw \mw_parse_filename_loop_test: }
```

The test `\token_if_expandable:NTF` should not be used here, since it returns false for undefined control sequences (by design). But when parsing a file name, undefined control sequences would be expanded, and cause errors. In order to keep this behaviour, we do the test ourselves.

A non-expandable token is either a non-space character, a space character, or a control sequence (either a primitive or a register).

```
\cs_new_protected_nopar:Npn \mw_parse_filename_loop_test:
  {
    \exp_after:wN \if_meaning:w \exp_not:N \g_peek_token \g_peek_token
      \exp_after:wN \use_ii:nn
    \else:
      \exp_after:wN \use_i:nn
    \fi:
      { \exp_after:wN \mw_parse_filename_loop:w }
      {
        \token_if_cs:NTF \g_peek_token
          { \mw_parse_filename_end: }
          {
            \mw_parse_filename_space_test:NTF \g_peek_token
              { \mw_parse_filename_space_do:w }
              {
                \exp_after:wN \mw_parse_filename_push:w
                \token_to_meaning:N
              }
          }
      }
  }
```

We stop at the first non-expandable non-character token.

```
\cs_new_protected_nopar:Npn \mw_parse_filename_end:
  {
    \group_align_safe_end:
    \group_end:
  }
```

We also stop when seeing a space, after removing it.

```
 %^^A TeXlive's behaviour (?): space with any catcode
\cs_new_protected_nopar:Npn \mw_parse_filename_space_test:NTF
  { \token_if_eq_charcode:NNTF \c_space_token }
\cs_new_protected_nopar:Npn \mw_parse_filename_space_do:w
  {
    \tex_afterassignment:D \mw_parse_filename_end:
    \cs_set_eq:NN \g_peek_token
  }
```

Finally, the case of normal characters. It seems that e.g., `\bgroup` is seen in the same way as an explicit left-brace character. Hence, we work with the `\meaning` of the character we are looking at. Most meanings of characters are two words, followed by the character. Three exceptions: "math shift character", "alignment tab character", and "macro parameter character", which have three words.

```
\cs_new_protected_nopar:Npn \mw_parse_filename_push:w #1 ~ %
  {
    \tl_if_in:nnTF { math ~ alignment ~ macro } { #1 }
      { \mw_parse_filename_push_three:w }
      { \mw_parse_filename_push_two:w }
  }
\cs_new_protected_nopar:Npn \mw_parse_filename_push_two:w #1 ~ #2
  {
    \tl_gput_right:Nn \g_mw_filename_tl {#2}
    \mw_parse_filename_loop:w
  }
\cs_new_protected_nopar:Npn \mw_parse_filename_push_three:w #1 ~ #2 ~ #3
  {
    \tl_gput_right:Nn \g_mw_filename_tl {#3}
    \mw_parse_filename_loop:w
  }
```

(*End definition for* `\mw_parse_filename:Nw`.)

3.3 Immediate (writing)

In the context of immediate writing, we can store the text in a token list, and only write it at the corresponding `\closeout` command. We keep track of a property list, `\g_mw_writes_prop`, of the writes which are open (from the point of view of the user), with the corresponding file name.

3.3.1 What follows `\immediate`

`\mw_prg_case_meaning:Nnn`
`\mw_prg_case_meaning_aux:Nw`

See `\prg_case_tl:Nnn`. Here, it is used by `\mw_immediate:w` to decide whether the following token is a writing command.

```
\cs_new:Npn \mw_prg_case_meaning:Nnn #1#2#3
  { \mw_prg_case_meaning_aux:Nw #1 #2 #1 {#3} \q_recursion_stop }
\cs_new:Npn \mw_prg_case_meaning_aux:Nw #1#2#3
  {
    \token_if_eq_meaning:NNTF #1 #2
      { \prg_case_end:nw {#3} }
      { \mw_prg_case_meaning_aux:Nw #1 }
  }
```

(*End definition for* `\mw_prg_case_meaning:Nnn`.)

`\mw_exp_after:Nf` Eventually, this should be replaced by a slow `\futurelet`-based expansion which does not lose a space.

```
\cs_new:Npn \mw_exp_after:Nf #1
  { \exp_after:wN #1 \tex_romannumeral:D -`0 }
```

(*End definition for* `\mw_exp_after:Nf`.)

`\mw_immediate:w` This is a little bit subtle: TEX's `\immediate` primitive raises a flag which is cancelled once TEX sees a non-expandable token. This can be emulated by expanding tokens in front of it until the first non-expandable token, and checking if it is either `\openout`, `\write`, or `\closeout`. We will use f-type expansion.[1]

```
\cs_new_protected_nopar:Npn \mw_immediate:w
  {
    \scan_stop: %^^A?
    \mw_exp_after:Nf \mw_immediate_aux:w
  }
\cs_new_protected_nopar:Npn \mw_immediate_aux:w
  {
    \peek_meaning:NT \mw_scan_stop:
      { \mw_immediate_mw_scan_stop:N }
  }
\cs_new_protected_nopar:Npn \mw_immediate_mw_scan_stop:N #1
  {
    \str_if_eq:nnTF { #1 } { \mw_scan_stop: }
      { \mw_immediate_do:NN }
      { #1 }
  }
\cs_new_protected_nopar:Npn \mw_immediate_do:NN #1 #2
  {
    \exp_args:Nc \mw_parse_number:Nw
      {
        \exp_after:wN \mw_immediate_do_aux:w
        \token_to_str:N #2
      }
```

[1] Unfortunately, this is not quite correct, since it loses a space if someone puts `\immediate` at places where it does not belong, such as `\immediate \c_space_tl`.

```
  }
\use:x
  {
    \cs_new_protected_nopar:Npn \exp_not:N \mw_immediate_do_aux:w
      ##1 \tl_to_str:n {mw_} { mw_immediate_ }
  }
```

(*End definition for* `\mw_immediate:w`.)

3.3.2 Immediate closeout

`\mw_immediate_closeout_test:` When the user requests to close a stream, we look in `\g_mw_reserved_writes_seq` to see if it is a reserved stream: in this case, we simply use the primitive.

```
\cs_new_protected_nopar:Npn \mw_immediate_closeout_test:
  {
    \seq_if_in:NVTF \g_mw_reserved_writes_seq \g_mw_stream_int
      { \mw_tex_immediate:w \mw_tex_closeout:w \g_mw_stream_int }
      { \mw_immediate_closeout_aux: }
  }
```

(*End definition for* `\mw_immediate_closeout_test:`.)

`\mw_immediate_closeout_aux:` We then look in `\g_mw_writes_prop` to find the file name corresponding to that stream number. If the stream does not appear as a key in the property list, then it was not open yet, and we do nothing.

```
\cs_new_protected_nopar:Npn \mw_immediate_closeout_aux:
  {
    \exp_args:NNV \prop_pop:NnNT
        \g_mw_writes_prop \g_mw_stream_int \l_mw_tmpa_tl
      {
        \mw_tex_immediate:w \mw_tex_openout:w
            \g_mw_write \l_mw_tmpa_tl \scan_stop:
        \group_begin:
          \int_set_eq:NN \tex_newlinechar:D \c_minus_one
          \tl_use:c { g_mw_write_ \int_use:N \g_mw_stream_int _tl }
          \tl_gclear:c { g_mw_write_ \int_use:N \g_mw_stream_int _tl }
        \group_end:
        \mw_tex_immediate:w \mw_tex_closeout:w \g_mw_write
      }
  }
```

(*End definition for* `\mw_immediate_closeout_aux:`.)

3.3.3 Immediate openout

`\mw_immediate_openout_test:` Read the stream number. If it is one of the reserved streams, we use the primitive. Otherwise, parse an optional equal sign, followed by the file name.

```
\cs_new_protected_nopar:Npn \mw_immediate_openout_test:
  {
    \seq_if_in:NVTF \g_mw_reserved_writes_seq \g_mw_stream_int
      { \mw_tex_immediate:w \mw_tex_openout:w \g_mw_stream_int }
      { \mw_parse_equals_filename:Nw \mw_immediate_openout_aux: }
  }
```

(*End definition for* `\mw_immediate_openout_test:`.)

`\mw_immediate_openout_aux:` When the user requests to open a stream, it might already be open, with another file as its destination. We thus need to first close the stream, writing all that we collected so far to that other file. This has no effect if the stream was not open yet.

We then put the stream and its associated filename in the property list, and empty the corresponding token list.

```
\cs_new_protected_nopar:Npn \mw_immediate_openout_aux:
  {
    \mw_immediate_closeout_aux:
    \prop_gput:NVV \g_mw_writes_prop \g_mw_stream_int \g_mw_filename_tl
    \tl_gclear_new:c { g_mw_write_ \int_use:N \g_mw_stream_int _tl }
  }
```

(*End definition for* `\mw_immediate_openout_aux:`.)

3.3.4 Immediate write

`\mw_immediate_write_test:` Read the stream number. If it is one of the reserved streams, we use the primitive. Otherwise, parse the text.

```
\cs_new_protected_nopar:Npn \mw_immediate_write_test:
  {
    \seq_if_in:NVTF \g_mw_reserved_writes_seq \g_mw_stream_int
      { \mw_tex_immediate:w \mw_tex_write:w \g_mw_stream_int }
      { \mw_parse_text:Nw \mw_immediate_write_aux: }
  }
```

(*End definition for* `\mw_immediate_write_test:`.)

`\mw_immediate_write_aux:` Only `\write` itself can emulate how `\write` expands tokens, because `#` don't have to be doubled. Hence, we write to a file, and re-read from it, setting `\endlinechar` to -1 to avoid spurious characters. Lines will be written one at a time: this allows us to set `\newlinechar` to -1 when writing, so that no extra line will be created.

If the stream `\g_mw_stream_int` is not allocated, then write either to the terminal or only to the log file.

```
\cs_new_protected_nopar:Npn \mw_immediate_write_aux:
  {
    \prop_if_in:NVTF \g_mw_writes_prop \g_mw_stream_int
```

```
      {
        \mw_tex_immediate:w \mw_tex_openout:w \g_mw_write
          \c_mw_tmp_file_tl \scan_stop:
        \mw_tex_immediate:w \mw_tex_write:w \g_mw_write { \g_mw_text_tl }
        \mw_tex_immediate:w \mw_tex_closeout:w \g_mw_write
        \group_begin:
          \int_set_eq:NN \tex_endlinechar:D \c_minus_one
          \mw_tex_openin:w \g_mw_read \c_mw_tmp_file_tl \scan_stop:
          \mw_immediate_write_aux_readlines:
          \mw_tex_closein:w \g_mw_read
        \group_end:
      }
      {
        \mw_tex_immediate:w \mw_tex_write:w
          \if_num:w \g_mw_stream_int < \c_zero
            -1
          \else:
            16
          \fi:
          { \g_mw_text_tl }
      }
  }
\cs_new_protected_nopar:Npn \mw_immediate_write_aux_readlines:
  {
    \mw_etex_readline:w \g_mw_read to \l_tmpa_tl
    \ior_if_eof:NF \g_mw_read
      {
        \tl_gput_right:cx
          { g_mw_write_ \int_use:N \g_mw_stream_int _tl }
          { \mw_tex_immediate:w \mw_tex_write:w \g_mw_write { \l_tmpa_tl } }
        \mw_immediate_write_aux_readlines:
      }
  }
```

(*End definition for* `\mw_immediate_write_aux:`.)

3.4 Non-immediate writing

This is trickier, because the expansion of the text for a non-immediate `\write` takes place immediately after the page containing it is shipped out. We store each non-immediate `\openout`, `\write`, or `\closeout` without expansion in separate token lists `\g_mw_late_write_`⟨*N*⟩`_tl` to be used later, and instead write `‘(`⟨*N*⟩`)` to a file (including the strange delimiters). After each shipout, we can read the file to see which output operations we need to perform, and in what order.

`\mw_late_aux:nN` Store the action to be done at shipout in a token list, and non-immediately write the label `\g_mw_late_write_int` of the output operation to the temporary file. Here, `#1`

holds an assignment similar to the lines above it, and `#2` holds the relevant immediate action to be performed after shipout.

```
\cs_new_protected_nopar:Npn \mw_late_aux:nN #1 #2
  {
    \int_gincr:N \g_mw_late_write_int
    \tl_const:cx { c_mw_late_write_ \int_use:N \g_mw_late_write_int _tl }
      {
        \int_gset:Nn \exp_not:N \g_mw_stream_int
          { \exp_not:V \g_mw_stream_int }
        #1
        \exp_not:N #2
      }
    \exp_args:NNx \mw_tex_write:w \g_mw_write
      { `( \int_use:N \g_mw_late_write_int ) }
  }
```

(*End definition for* `\mw_late_aux:nN`.)

`\mw_openout:w` `\mw_openout_test:` `\mw_openout_aux:`

`\openout` tests if the number to come is among reserved streams. If it is, use the primitive, otherwise, parse a filename.

```
\cs_new_protected_nopar:Npn \mw_openout:w
  { \mw_scan_stop: \mw_parse_number:Nw \mw_openout_test: }
\cs_new_protected_nopar:Npn \mw_openout_test:
  {
    \seq_if_in:NVTF \g_mw_reserved_writes_seq \g_mw_stream_int
      { \mw_tex_openout:w \g_mw_stream_int }
      { \mw_parse_equals_filename:Nw \mw_openout_aux: }
  }
\cs_new_protected_nopar:Npn \mw_openout_aux:
  {
    \mw_late_aux:nN
      {
        \tl_gset:Nn \exp_not:N \g_mw_filename_tl
          { \exp_not:V \g_mw_filename_tl }
      }
      \mw_immediate_openout_aux:
  }
```

(*End definition for* `\mw_openout:w`.)

`\mw_write:w` `\mw_write_test:` `\mw_write_aux:`

Same idea for `\write`, except that we parse a text.

```
\cs_new_protected_nopar:Npn \mw_write:w
  { \mw_scan_stop: \mw_parse_number:Nw \mw_write_test: }
\cs_new_protected_nopar:Npn \mw_write_test:
  {
    \seq_if_in:NVTF \g_mw_reserved_writes_seq \g_mw_stream_int
      { \mw_tex_write:w \g_mw_stream_int }
```

```
      { \mw_parse_text:Nw \mw_write_aux: }
  }
\cs_new_protected_nopar:Npn \mw_write_aux:
  {
    \mw_late_aux:nN
      {
        \tl_gset:Nn \exp_not:N \g_mw_text_tl
          { \exp_not:V \g_mw_text_tl }
      }
      \mw_immediate_write_aux:
  }
```

(*End definition for* `\mw_write:w`.)

`\mw_closeout:w` `\mw_closeout_test:` `\mw_closeout_aux:`

Same idea for `\closeout`, and we don't need to parse anything else than the number.

```
\cs_new_protected_nopar:Npn \mw_closeout:w
  { \mw_scan_stop: \mw_parse_number:Nw \mw_closeout_test: }
\cs_new_protected_nopar:Npn \mw_closeout_test:
  {
    \seq_if_in:NVTF \g_mw_reserved_writes_seq \g_mw_stream_int
      { \mw_tex_closeout:w \g_mw_stream_int }
      { \mw_closeout_aux: }
  }
\cs_new_protected_nopar:Npn \mw_closeout_aux:
  { \mw_late_aux:nN { } \mw_immediate_closeout_aux: }
```

(*End definition for* `\mw_closeout:w`.)

`\mw_before_shipout:`

Immediately before the shipout, we must open the writing stream `\g_mw_write`. Each delayed output operation has been replaced by `\write \g_mw_write {‘(`⟨*operation number*⟩`}`. The delimiters we chose to put around numbers must be at least two distinct characters on the left (then `\tex_newlinechar:D` cannot be equal to the delimiter), and at least one non-digit character on the right.

```
\cs_new_protected_nopar:Npn \mw_before_shipout:
  {
    \mw_tex_immediate:w \mw_tex_openout:w \g_mw_write
      \c_mw_tmp_file_tl \scan_stop:
  }
```

(*End definition for* `\mw_before_shipout:`.)

`\mw_after_shipout:`

After all the `\write`s are performed, we read the file, and grab the relevant pieces in a seq.

```
\seq_new:N \g_mw_operations_seq
\cs_new_protected_nopar:Npn \mw_after_shipout:
  {
    \mw_tex_immediate:w \mw_tex_closeout:w \g_mw_write
```

```
    \group_begin:
      \int_set_eq:NN \tex_endlinechar:D \tex_newlinechar:D
      \tl_map_inline:nn { `(0123456789) }
        { \char_set_catcode_other:n {`##1} }
      \etex_everyeof:D { `() \exp_not:N }
      \tl_gset:Nx \g_mw_operations_seq
        { \if_false: } \fi:
          \exp_after:wN \mw_after_shipout_loop:ww
          \tex_input:D \c_mw_tmp_file_tl \scan_stop:
    \group_end:
    \seq_map_inline:Nn \g_mw_operations_seq
      {
        \tl_use:c
          { c_mw_late_write_ ##1 _tl }
      }
  }
\cs_new:Npn \mw_after_shipout_loop:ww #1 `( #2 )
  {
    \tl_if_empty:nTF {#2}
      { \if_false: { \fi: } }
      {
        \exp_not:N \seq_item:n {#2}
        \mw_after_shipout_loop:ww
      }
  }
```

(*End definition for* `\mw_after_shipout:`.)

`\shipout` `\mw_shipout:w`

If atbegshi is available, patch it by adding `\mw_after_shipout:` at the right place. Otherwise, redefine `\shipout` to add a hook.

```
\IfFileExists{atbegshi.sty}
  {
    \RequirePackage{atbegshi}
    \tl_replace_in:Nnn \AtBegShi@Output
      { \AtBegShi@OrgShipout \box \AtBeginShipoutBox }
      {
        \mw_before_shipout:
        \AtBegShi@OrgShipout \box \AtBeginShipoutBox
        \mw_after_shipout:
      }
    \tl_replace_in:Nnn \AtBegShi@Output
      { \AtBeginShipoutOriginalShipout \box \AtBeginShipoutBox }
      {
        \mw_before_shipout:
        \AtBeginShipoutOriginalShipout \box \AtBeginShipoutBox
        \mw_after_shipout:
      }
  }
  {
```

```
  \cs_new_protected_nopar:Npn \mw_shipout:w
    {
      \int_gset_eq:NN \g_mw_tmpa_int \etex_currentgrouplevel:D
      \tex_afterassignment:D \mw_shipout_aux:
      \tex_global:D \tex_setbox:D \g_mw_box
    }
  \cs_new_protected_nopar:Npn \mw_shipout_aux:
    {
      \int_compare:nNnTF { \g_mw_tmpa_int }
                         = { \etex_currentgrouplevel:D }
        { \mw_shipout_aux_ii: }
        { \tex_aftergroup:D \mw_shipout_aux_ii: }
    }
  \cs_new_protected_nopar:Npn \mw_shipout_aux_ii:
    {
      \mw_before_shipout:
      \mw_tex_shipout:w \tex_box:D \g_mw_box
      \mw_after_shipout:
    }
  \cs_gset_eq:NN \shipout \mw_shipout:w
}
```

(*End definition for* `\shipout`.)

3.5 Hook at the very end

`\mw_close_all:` At the end of the document, close all the files.

```
\cs_new_protected_nopar:Npn \mw_close_all:
  {
    \prop_map_inline:Nn \g_mw_writes_prop
      {
        \mw_tex_immediate:w \mw_tex_openout:w \g_mw_write ##2 \scan_stop:
        \group_begin:
          \int_set_eq:NN \tex_newlinechar:D \c_minus_one
          \tl_use:c { g_mw_write_ ##1 _tl }
          \tl_gclear:c { g_mw_write_ ##1 _tl }
        \group_end:
        \mw_tex_immediate:w \mw_tex_closeout:w \g_mw_write
      }
    \prop_gclear:N \g_mw_writes_prop
  }
```

(*End definition for* `\mw_close_all:`.)

`\mw_put_at_end:Nw` This pushes its first argument to the very end of the LaTeX run, recursively, just in case some other code adds things there. If atveryend is available, we use it instead.

```
\IfFileExists{atveryend.sty} { \use_i:nn } { \use_ii:nn }
```

```
{
  \RequirePackage {atveryend}
  \AtVeryVeryEnd { \mw_close_all: }
}
{
  \cs_new_protected:Npn \mw_put_at_end:Nw #1 #2 \@@end
    {
      \tl_if_empty:nTF {#2}
        { #1 \@@end }
        { #2 \mw_put_at_end:Nw #1 \@@end }
    }
  \AtEndDocument { \mw_put_at_end:Nw \mw_close_all: }
}
```

(*End definition for* `\mw_put_at_end:Nw`.)

3.6 Modified `\newwrite`

`\newwrite` We need to allow `\newwrite` to allocate more than 16 writes, but beware that 18 is reserved, and that packages might expect 16 or 17 to write to the terminal. So instead skip until 20.

```
\countdef \mw_write_int 17 \scan_stop:
\cs_new:Npn \mw_newwrite:N #1
  {
    \if_num:w \mw_write_int = \c_sixteen
      \int_gset:Nn \mw_write_int { 20 }
    \fi:
    \int_set_eq:NN \allocationnumber \mw_write_int
    \int_gincr:N \mw_write_int
    \cs_undefine:N #1
    \int_const:Nn #1 { \allocationnumber }
    \wlog {\string #1=\string \write \the \allocationnumber }
  }
```

(*End definition for* `\newwrite`.)

3.7 Redefining the "normal" control sequences

`\immediate` `\openout` `\write` `\closeout` `\newwrite` `\shipout` has been redefined earlier.

```
\cs_gset_eq:NN \immediate \mw_immediate:w
\cs_gset_eq:NN \openout   \mw_openout:w
\cs_gset_eq:NN \write     \mw_write:w
\cs_gset_eq:NN \closeout  \mw_closeout:w
\cs_gset_eq:NN \newwrite  \mw_newwrite:N
```

(*End definition for* `\immediate` *and others.*)

</package>