The primargs package: Parsing arguments of primitives

Bruno Le Floch

2017/04/10

Contents

1 primargs documentation

This TEX and LATEX package is currently used by morewrites when redefining primitives: it allows to read arguments of primitives in place of TEX, which is useful to add hooks to primitives. Of course, this is much slower than letting TEX do things directly.

All assignments done by this package are global. While a negative value of the `\globaldefs` (primitive) parameter normally makes all assignments local, this package makes sure `\globaldefs` is non-negative before assignments.

1.1 Reading one token without removing it

`\g_primargs_token`

The token read by `\primargs_read_token:N` or `\primargs_read_x_token:N`. Its value is always set globally. It can be an `\outer` macro.

\primargs_read_token:N `\primargs_read_token:N` ⟨*function*⟩

Sets `\g_primargs_token` equal to the token following the ⟨*function*⟩, then calls the ⟨*function*⟩. The token following the ⟨*function*⟩ is not removed.

TEXhackers note: This is essentially `\global \futurelet \g_primargs_token` ⟨*function*⟩, with the added guarantee that the assignment is global even when `\globaldefs` is negative.

\primargs_read_x_token:N `\primargs_read_x_token:N` ⟨*function*⟩

Expands tokens recursively with `\exp_after:wN` until encountering a non-expandable token and afterwards calls the ⟨*function*⟩. The non-expandable token following the ⟨*function*⟩ is not removed and `\g_primargs_token` is also set (globally) equal to that token.

1.2 Removing tokens

\primargs_remove_token:N `\primargs_remove_token:N` ⟨*function*⟩

Removes the ⟨*token*⟩ which follows the ⟨*function*⟩, then calls the ⟨*function*⟩. This also sets `\g_primargs_token` (globally) equal to the removed token.

\primargs_remove_one_optional_space:N `\primargs_remove_one_optional_space:N` ⟨*function*⟩

Expands tokens following the ⟨*function*⟩ until a non-expandable token is found, and sets `\g_primargs_token` (globally) equal to this token, then removes the token if it has catcode 10 (space). Finally, call the ⟨*function*⟩.

\primargs_remove_optional_spaces:N `\primargs_remove_optional_spaces:N` ⟨*function*⟩

Expands tokens following the ⟨*function*⟩, removing any token with catcode 10 (space), then sets `\g_primargs_token` (globally) equal to the first non-space token and calls the ⟨*function*⟩.

\primargs_remove_equals:N `\primargs_remove_equals:N` ⟨*function*⟩

Expands tokens following the ⟨*function*⟩, removing any token with catcode 10 (space), then sets `\g_primargs_token` (globally) equal to the first non-space token. If this token is an explicit `=` character token with catcode 12 (other), then it is removed as well. Finally, calls the ⟨*function*⟩.

\primargs_remove_filler:N `\primargs_remove_filler:N` ⟨*function*⟩

New: 2014-08-06

Expands tokens following the ⟨*function*⟩, removing any token with catcode 10 (space) or equal to `\relax`, then sets `\g_primargs_token` (globally) equal to the next token. Finally, calls the ⟨*function*⟩.

1.3 Grabbing arguments

\primargs_get_number:N
\primargs_get_dimen:N
\primargs_get_glue:N
\primargs_get_mudimen:N
\primargs_get_muglue:N

`\primargs_get_number:N` ⟨*function*⟩

Reads a number/dimension/glue/math dimension/math glue following the ⟨*function*⟩, then calls the ⟨*function*⟩ with a braced argument containing the value found. For instance,

```
\primargs_get_glue:N \test 3sp plus \numexpr 2-3 fill X
```

yields

```
\test {3sp plus -1fill}X
```

A word of warning: the `\primargs_get_mudimen:N` function currently parses a ⟨*muskip*⟩ instead of a ⟨*mudimen*⟩.

\primargs_get_general_text:N

`\primargs_get_general_text:N` ⟨*function*⟩

Updated: 2014-08-06

Finds what TEX's grammar calls a ⟨*general text*⟩ (that is, a ⟨*filler*⟩, a catcode 1 token, a ⟨*balanced text*⟩, and an explicit catcode 2 token) following the ⟨*function*⟩, and calls the ⟨*function*⟩ with the ⟨*balanced text*⟩ as a braced argument.

\primargs_get_file_name:N
\primargs_get_input_file_name:N

`\primargs_get_file_name:N` ⟨*function*⟩
`\primargs_get_input_file_name:N` ⟨*function*⟩

Updated: 2017-04-10

Reads a ⟨*file name*⟩ following the ⟨*function*⟩ and calls the ⟨*function*⟩ with this ⟨*file name*⟩ as a braced argument. The two functions are identical except in the LuaTEX engine where `\primargs_get_input_file_name:N` allows braced file names: LuaTEX allows such braced file names for some primitives (`\input` and `\openin`) but not others (`\openout`).

TEXhackers note: When braced file names are disallowed, the file name is obtained by discarding ⟨*optional spaces*⟩ then repeatedly doing the following. Fully expand what follows in the input stream. If the next token is an explicit or implicit character token (regardless of its catcode) then add that character to the file name and remove it from the input stream, and go back to expanding tokens, except in one case: if the character code is 32 (space) and the number of quote characters (code 34) already in the file name is even, then the space is removed from the input stream, not included in the file name, and parsing ends. Finally, if the next token is a non-expandable command (be it a control sequence or an active character) then the file name ends and the command is left in the input stream.

When braced file names are allowed, the following steps are added prior to the procedure above. First remove a ⟨*filler*⟩. If the next token is of catcode 1 then fully expand tokens one by one and add their string representation (with `\tl_to_str:n`, not `\token_to_str:N`) to the file name.

1.4 Comments and internal functions

This package is not idiomatic expl3 and should not be used as an example of good coding practices. It uses `\...:D` primitives directly:

- to cope with `\outer` tokens, since this package is meant to be used quite broadly;
- for primitives with (rightfully) no expl3 interface (or a slightly incomplete interface), namely `\afterassignment`, `\globaldefs`, `\aftergroup`, `\the`, `\deadcycles`, `\hoffset`, `\topskip`, `\thinmuskip`, `\unexpanded`;
- to test that a token's meaning is a given primitive when the expl3 interface is not (or not obviously) a copy of the primitive.

As a result, *do not take this package as an example of how to code with expl3; go and see Joseph Wright's siunitx for instance.*

`\__primargs_get_rhs:NnN`
`\__primargs_get_rhs:NoN`

`\__primargs_get_rhs:NnN` ⟨*register*⟩ {⟨*register rhs*⟩} ⟨*function*⟩

Use the ⟨*register*⟩ to find a right-hand side of a valid assignment for this type of variable, and feed the value found to the ⟨*function*⟩. The value of the ⟨*register*⟩ is then restored using ⟨*register*⟩ = ⟨*register rhs*⟩, where the ⟨*register rhs*⟩ should be the initial value of the ⟨*register*⟩. All those assignments are performed within a group, but some are automatically global, and `\globaldefs` may cause trouble with others.

Despite large efforts expended to make this package robust against changes to the `\globaldefs` parameter, setting it to a non-zero value may make some parts of this package crash.

Tokens inserted using `\afterassignment` may be lost when using this package, since it uses `\afterassignment` internally.

Todo list.

- Test all functions within alignments and understand their interaction with the master counter.
- Correct the parsing of ⟨*mudimen*⟩.
- Perhaps parse ⟨*muglue*⟩ and ⟨*glue*⟩ by hand to avoid bad interactions with `\globaldefs`. Otherwise put up a warning about `\globaldefs` when relevant. Better partial fix: declare a skip and a muskip.
- Write tests of engine behaviour, especially LuaTeX's `\input`, `\openin`, `\openout` including behaviour of `#` and spaces and character-code-zero, to detect unexpected changes. In `\input{`...`\input`...`}`, LuaTeX expands the inner `\input` but uses the inner file name as the outer file name.

2 primargs implementation

<*package>

```
\RequirePackage {expl3} [2017/03/18]
\ProvidesExplPackage
  {primargs} {2017/04/10} {} {Parsing arguments of primitives}
```

⟨@@=primargs⟩

2.1 Variables and helpers

\g__primargs_code_tl Used to contain temporary code.

```
\tl_new:N \g__primargs_code_tl
```

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

\g__primargs_file_name_tl
\g__primargs_file_name_level_tl
Token list used to build a file name, one character at a time. Token list holding the level of nesting in quotes or braces.

```
\tl_new:N \g__primargs_file_name_tl
\tl_new:N \g__primargs_file_name_level_tl
```

(*End definition for* `\g__primargs_file_name_tl` *and* `\g__primargs_file_name_level_tl`.)

__primargs_safe: This function, which must be called in a group, cancels any `\afterassignment` token and makes the `\globaldefs` parameter non-negative. This ensures that assignments prefixed with `\global` are indeed global. When `\globaldefs` is positive, every assignment is global, and it is not possible to safely (locally) set it to zero.

```
\cs_new_protected:Npn \__primargs_safe:
  {
    \tex_afterassignment:D \tex_relax:D
    \if_int_compare:w 0 > \tex_globaldefs:D
      \int_zero:N \tex_globaldefs:D
    \fi:
  }
```

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

2.2 Read token with or without expansion

TEX often calls the `get_x_token` procedure when parsing various parts of its grammar. This expands tokens recursively until reaching a non-expandable token. We emulate this by reading the next token with `\futurelet`, checking whether it is expandable or not by comparing its meaning to its meaning when acted upon by `\noexpand`, and expanding it with `\expandafter` if it is expandable.

One thing to be careful about is that

```
\expandafter \show \noexpand \space
```

shows the `\meaning` of the `\notexpanded: \space`, namely `\relax` (frozen, in fact, hence a bit different from the normal `\relax`), while expanding twice with

```
\expandafter \expandafter \expandafter \show \noexpand \space
```

expands the `\space` to the underlying space character token. What this means is that we must first check if the token is expandable or not, and only then expand, and that the token should not be queried again using `\futurelet`. On this latter point, run

```
\def \test { \show \next \futurelet \next \test }
\expandafter \test \noexpand \space
```

to see how `\next` changes from `\relax` to becoming a macro.

\primargs_read_x_token:N
__primargs_read_x_token:N
__primargs_read_x_token_aux:N
__primargs_read_x_token_std:N
__primargs_read_x_token_file:N

This is a bit messy, because we need to support the fact that TEX does not consider `\input` as expandable when it is looking for a file name. This variation is encapsulated by letting `\__primargs_read_x_token_aux:N` equal to either a standard (`std`) version or a version specific to file names (`file`).

First query the following token. Then test whether it is expandable, using a variant of the `\token_if_expandable:NTF` test.[1] If the token is expandable, `\exp_not:N` will change its `\meaning` to `\relax`, the test is `false`, we expand, and call the loop. Otherwise, we stop. In the `file` version there is an extra test for `\tex_input:D`. By default use the standard version.

```
\cs_new_protected:Npn \primargs_read_x_token:N
  {
    \group_begin:
      \__primargs_safe:
      \__primargs_read_x_token:N
  }
\cs_new_protected:Npn \__primargs_read_x_token:N
  {
    \tex_afterassignment:D \__primargs_read_x_token_aux:N
    \tex_global:D \tex_futurelet:D \g_primargs_token
  }
\cs_new_protected:Npn \__primargs_read_x_token_std:N
  {
    \exp_after:wN
    \if_meaning:w \exp_not:N \g_primargs_token \g_primargs_token
      \group_end: \use_i:nnnn
    \fi:
    \exp_after:wN \__primargs_read_x_token:N \exp_after:wN
  }
\cs_new_eq:NN \__primargs_read_x_token_aux:N
              \__primargs_read_x_token_std:N
\cs_new_protected:Npn \__primargs_read_x_token_file:N
  {
    \if_meaning:w \tex_input:D \g_primargs_token
      \use_i_ii:nnn \group_end:
    \fi:
    \__primargs_read_x_token_std:N
  }
```

(*End definition for* `\primargs_read_x_token:N` *and others. These functions are documented on page* 2.)

\primargs_read_token:N

The same without expansion, useful for instance when we already know that what follows is expanded. Interestingly, we don't ever need to take the user's function as an argument.

```
\cs_new_protected:Npn \primargs_read_token:N
  {
    \group_begin:
      \__primargs_safe:
      \tex_afterassignment:D \group_end:
      \tex_global:D \tex_futurelet:D \g_primargs_token
  }
```

(*End definition for* `\primargs_read_token:N`. *This function is documented on page* 2.)

[1] This LaTeX3 test returns `false` for undefined tokens (by design), but TEX's `get_x_token` expands those undefined tokens, causing errors, so we should as well.

2.3 Removing tokens

\primargs_remove_token:N

Remove token using `\let` (note the presence of `=` and a space, to correctly remove explicit space characters), then insert the ⟨*function*⟩ after closing the group.

```
\cs_new_protected:Npn \primargs_remove_token:N #1
  {
    \group_begin:
      \__primargs_safe:
      \tex_aftergroup:D #1
      \tex_afterassignment:D \group_end:
      \tex_global:D \tex_let:D \g_primargs_token = ~
  }
```

(*End definition for* `\primargs_remove_token:N`*. This function is documented on page* 2.)

\primargs_remove_one_optional_space:N
__primargs_remove_one_optional_space:

Start a group: we will insert the ⟨*function*⟩ at its end.

```
\cs_new_protected:Npn \primargs_remove_one_optional_space:N #1
  {
    \group_begin:
      \__primargs_safe:
      \tex_aftergroup:D #1
      \primargs_read_x_token:N \__primargs_remove_one_optional_space:
  }
\cs_new_protected:Npn \__primargs_remove_one_optional_space:
  {
    \if_catcode:w \c_space_token \exp_not:N \g_primargs_token
      \exp_after:wN \primargs_remove_token:N
    \fi:
    \group_end:
  }
```

(*End definition for* `\primargs_remove_one_optional_space:N` *and* `\__primargs_remove_one_optional_space:`*. These functions are documented on page* 2.)

\primargs_remove_optional_spaces:N
__primargs_remove_optional_spaces:
__primargs_remove_optional_spaces_aux:

Start a group, make assignments safe, then recursively expand tokens and remove any token with catcode 10 (space). Once another token is found, close the group hence insert the ⟨*function*⟩ `#1`.

```
\cs_new_protected:Npn \primargs_remove_optional_spaces:N #1
  {
    \group_begin:
      \__primargs_safe:
      \tex_aftergroup:D #1
      \__primargs_remove_optional_spaces:
  }
\cs_new_protected:Npn \__primargs_remove_optional_spaces:
  { \primargs_read_x_token:N \__primargs_remove_optional_spaces_aux: }
\cs_new_protected:Npn \__primargs_remove_optional_spaces_aux:
  {
    \if_catcode:w \c_space_token \exp_not:N \g_primargs_token
      \exp_after:wN \primargs_remove_token:N
      \exp_after:wN \__primargs_remove_optional_spaces:
    \else:
      \exp_after:wN \group_end:
    \fi:
  }
```

(*End definition for* `\primargs_remove_optional_spaces:N`, `\__primargs_remove_optional_spaces:`, *and* `\__primargs_remove_optional_spaces_aux:`. *These functions are documented on page 2.*)

`\primargs_remove_equals:N`
`\__primargs_remove_equals:`
`\__primargs_remove_equals_aux:NN`

Remove ⟨*optional spaces*⟩, then test for an explicit `=`, both in `\meaning` and as a token list: once we know its `\meaning`, we can grab it safely.

```
\cs_new_protected:Npn \primargs_remove_equals:N #1
  {
    \group_begin:
      \tex_aftergroup:D #1
      \primargs_remove_optional_spaces:N \__primargs_remove_equals:
  }
\cs_new_protected:Npn \__primargs_remove_equals:
  {
      \if_meaning:w = \g_primargs_token
        \exp_after:wN \__primargs_remove_equals_aux:NN
      \fi:
    \group_end:
  }
\cs_new_protected:Npn \__primargs_remove_equals_aux:NN #1#2
  { \tl_if_eq:nnTF { #2 } { = } { #1 } { #1 #2 } }
```

(*End definition for* `\primargs_remove_equals:N`, `\__primargs_remove_equals:`, *and* `\__primargs_remove_equals_aux:NN`. *These functions are documented on page 2.*)

`\primargs_remove_filler:N`
`\__primargs_remove_filler:`
`\__primargs_remove_filler_aux:`
`\__primargs_remove_filler_end:NNNNN`

Within a group remove a ⟨*filler*⟩, and insert the user's `#1` after closing the group. A ⟨*filler*⟩ consists of tokens with catcode 10 (space) or equal to `\relax` or to the "frozen `\relax`" command.

```
\cs_new_protected:Npn \primargs_remove_filler:N #1
  {
    \group_begin:
      \__primargs_safe:
      \tex_aftergroup:D #1
      \__primargs_remove_filler:
  }
\cs_new_protected:Npn \__primargs_remove_filler:
  { \primargs_read_x_token:N \__primargs_remove_filler_aux: }
\cs_new_protected:Npn \__primargs_remove_filler_aux:
  {
    \if_catcode:w \c_space_token \exp_not:N \g_primargs_token
    \else:
      \if_meaning:w \tex_relax:D \g_primargs_token
      \else:
        \exp_after:wN
        \if_meaning:w \exp_not:N \prg_do_nothing: \g_primargs_token
        \else:
          \__primargs_remove_filler_end:NNNNN
        \fi:
      \fi:
    \fi:
    \primargs_remove_token:N \__primargs_remove_filler:
  }
\cs_new_protected:Npn \__primargs_remove_filler_end:NNNNN #1#2#3#4#5
  { #1 #2 #3 \group_end: }
```

(*End definition for* `\primargs_remove_filler:N` *and others. These functions are documented on page 2.*)

2.4 Right-hand sides of assignments

The naive approach to reading an integer, or a general text, is to let TEX perform an assignment to a `\count`, or a `\toks`, register and regain control using `\afterassignment`. The question is then to know which `\count` or `\toks` register to use. One might think that any can be used as long as the assignment happens in a group.

However, there comes the question of the `\globaldefs` parameter. If this parameter is positive, every assignment is global, including assignments to the parameter itself, preventing us from setting it to zero locally; hence, we are stuck with global assignments (if `\globaldefs` is negative, we can change it, locally, to whatever value pleases us, as done by `\__primargs_safe:`). We may thus not use scratch registers to parse integers, general texts, and other pieces of TEX's grammar.

For integers, we will use `\deadcycles`, a parameter which is automatically assigned globally, and we revert it to its previous value afterwards.

`\__primargs_get_rhs:NnN`
`\__primargs_get_rhs:NoN`

The last two lines of this function are the key: assign to `#1`, then take control using `\afterassignment`. After the assignment, we expand the value found, `\tex_the:D #1`, within a brace group, then restore `#1` using its initial value `#2`, and end the group. The earlier use of `\tex_aftergroup:D` inserts the ⟨*function*⟩ `#3` before the brace group containing the value found.

```
\cs_new_protected:Npn \__primargs_get_rhs:NnN #1#2#3
  {
    \group_begin:
      \__primargs_safe:
      \tex_aftergroup:D #3
      \tl_gset:Nn \g__primargs_code_tl
        {
          \use:x
            {
              \exp_not:n { #1 = #2 \group_end: }
              { \tex_the:D #1 }
            }
        }
      \tex_afterassignment:D \g__primargs_code_tl
      #1 =
  }
\cs_generate_variant:Nn \__primargs_get_rhs:NnN { No }
```

(*End definition for* `\__primargs_get_rhs:NnN`.)

`\primargs_get_number:N`

We use the general `\__primargs_get_rhs:NoN`, using the internal register `\deadcycles`, for which all assignments are global: thus, restoring its value will not interact badly with groups.

```
\cs_new_protected:Npn \primargs_get_number:N
  {
    \__primargs_get_rhs:NoN \tex_deadcycles:D
      { \tex_the:D \tex_deadcycles:D }
  }
```

(*End definition for* `\primargs_get_number:N`. *This function is documented on page* 3.)

\primargs_get_dimen:N

Use \hoffset as a register since it is not too likely to be changed locally (anyways, which register we use is not that important since normally, \globaldefs is zero, and everything is done within a group).

```
\cs_new_protected:Npn \primargs_get_dimen:N
  {
    \__primargs_get_rhs:NoN \tex_hoffset:D
      { \tex_the:D \tex_hoffset:D }
  }
```

(*End definition for* \primargs_get_dimen:N. *This function is documented on page* 3.)

\primargs_get_glue:N

Use \topskip.

```
\cs_new_protected:Npn \primargs_get_glue:N
  {
    \__primargs_get_rhs:NoN \tex_topskip:D
      { \tex_the:D \tex_topskip:D }
  }
```

(*End definition for* \primargs_get_glue:N. *This function is documented on page* 3.)

\primargs_get_mudimen:N

There is no such thing as a ⟨*mudimen variable*⟩, so we're on our own to parse a ⟨*mudimen*⟩. Warn about that problem, and parse a ⟨*muglue*⟩ instead.

```
\cs_new_protected:Npn \primargs_get_mudimen:N
  {
    \msg_warning:nn { primargs } { get-mudimen }
    \primargs_get_muglue:N
  }
\msg_new:nnn { primargs } { get-mudimen }
  { The~\iow_char:N\\primargs_get_mudimen:N~function~is~buggy. }
```

(*End definition for* \primargs_get_mudimen:N. *This function is documented on page* 3.)

\primargs_get_muglue:N

Use \thinmuskip.

```
\cs_new_protected:Npn \primargs_get_muglue:N
  {
    \__primargs_get_rhs:NoN \tex_thinmuskip:D
      { \tex_the:D \tex_thinmuskip:D }
  }
```

(*End definition for* \primargs_get_muglue:N. *This function is documented on page* 3.)

\primargs_get_general_text:N
__primargs_get_general_text:
__primargs_get_general_text_error:n

Getting a ⟨*general text*⟩ is more tricky, as an assignment to \errhelp (for instance) would also allow constructions such as \toks0. Instead, we remove a ⟨*filler*⟩ then test whether the next token (already expanded) is a catcode 1 token, in which case we replace it by an explicit left brace before calling the function. When the next token is not of catcode 1, we produce an error, attempting to imitate as closely as possible the TEX error.

```
\cs_new_protected:Npn \primargs_get_general_text:N #1
  {
    \group_begin:
      \__primargs_safe:
      \tex_aftergroup:D #1
      \tex_aftergroup:D { \if_false: } \fi:
      \primargs_remove_filler:N \__primargs_get_general_text:
  }
```

```
\cs_new_protected:Npn \__primargs_get_general_text:
  {
    \if_catcode:w \c_group_begin_token \g_primargs_token
      \exp_after:wN \primargs_remove_token:N
    \else:
      \group_begin:
        \tex_aftergroup:D \__primargs_get_general_text_error:n
        \if_catcode:w \c_group_end_token \g_primargs_token
          \tex_aftergroup:D {
          \tex_aftergroup:D }
        \fi:
    \fi:
    \group_end:
  }
\cs_new_protected:Npn \__primargs_get_general_text_error:n #1
  {
    \exp_after:wN \group_end:
    \etex_unexpanded:D \if_int_compare:w `{ = \c_zero \fi: #1 }
  }
```

(*End definition for* `\primargs_get_general_text:N`, `\__primargs_get_general_text:`, *and* `\__primargs_get_general_text_error:n`. *These functions are documented on page 3.*)

2.5 Get file name

`\primargs_get_file_name:N` Empty the file name (globally), and build it one character at a time. The ⟨*function*⟩ is added at the end of a group, started here. As described in the TEXbook, a ⟨*file name*⟩ should start with ⟨*optional spaces*⟩ (LuaTEX changes that to ⟨*filler*⟩), which we remove, then character tokens, ending with a non-expandable character or control sequence. After space removal, `\g_primargs_token` contains the next token, so no need for `\primargs_read_token:N`. When TEX reads a file name, the `\input` primitive is temporarily not expandable, so we temporarily change `\primargs_read_x_token:N` to not expand this primitive. This is reverted by `\__primargs_get_file_name_end:`.

```
\cs_new_protected:Npn \primargs_get_file_name:N #1
  {
    \group_begin:
      \__primargs_safe:
      \cs_gset_eq:NN \__primargs_read_x_token_aux:N
                     \__primargs_read_x_token_file:N
      \tex_aftergroup:D #1
      \tl_gclear:N \g__primargs_file_name_tl
      \tl_gset:Nn \g__primargs_file_name_level_tl { 0 }
      \primargs_remove_optional_spaces:N \__primargs_get_file_name_test:
  }
```

(*End definition for* `\primargs_get_file_name:N`. *This function is documented on page 3.*)

`\__primargs_get_file_name_test:` The token read is in `\g_primargs_token`, and is non-expandable. If it is a control sequence, end the ⟨*file name*⟩. Spaces are special (quotes too, but that is treated elsewhere). Otherwise, we extract the character from the `\meaning` of the ⟨*token*⟩, which we remove anyways: in that case, we'll recurse.

```
\cs_new_protected:Npn \__primargs_get_file_name_test:
  {
```

```
    \token_if_cs:NTF \g_primargs_token
      { \__primargs_get_file_name_end: }
      {
        \token_if_eq_charcode:NNTF \c_space_token \g_primargs_token
          { \primargs_remove_token:N \__primargs_get_file_name_space: }
          { \primargs_remove_token:N \__primargs_get_file_name_char: }
      }
  }
```

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

`\__primargs_get_file_name_end:` When the end of the file name is reached, reinstate the original definition of `read_x_token` so as to make `\input` expandable again, then end the group, after expanding the contents of `\g__primargs_file_name_tl`.

```
\cs_new_protected:Npn \__primargs_get_file_name_end:
  {
    \cs_gset_eq:NN \__primargs_read_x_token_aux:N
                   \__primargs_read_x_token_std:N
    \exp_args:No \group_end: \g__primargs_file_name_tl
  }
```

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

`\__primargs_get_file_name_space:` We have already removed the space from the input stream. If there is an odd number of quotes so far, add a space to the file name and continue. Otherwise the file name ends.

```
\cs_new_protected:Npn \__primargs_get_file_name_space:
  {
    \int_if_odd:nTF { \g__primargs_file_name_level_tl }
      {
        \tl_gput_right:Nn \g__primargs_file_name_tl { ~ }
        \primargs_read_x_token:N \__primargs_get_file_name_test:
      }
      { \__primargs_get_file_name_end: }
  }
```

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

`\__primargs_get_file_name_char:` `\__primargs_get_file_name_char_ii:w` `\__primargs_get_file_name_char_iii:w` Check for a quote, which switches `\g__primargs_file_name_level_tl` from 0 to 1 or back. With an explicit character, applying `\string` would give the character code. Here, implicit characters have to be converted too, so we must work with the `\meaning`, which is two or three words separated by spaces, then the character. The `ii` auxiliary removes the first two words, and duplicates the remainder (either one character, or a word and a character), and the second auxiliary leaves the second piece in the definition (in both cases, the character). Then loop with expansion. This technique would fail if the character could be a space (character code 32).

```
\cs_new_protected:Npn \__primargs_get_file_name_char:
  {
    \token_if_eq_charcode:NNT " \g_primargs_token
      {
        \tl_set:Nx \g__primargs_file_name_level_tl
          { \int_eval:n { 1 - \g__primargs_file_name_level_tl } }
      }
    \tl_gput_right:Nx \g__primargs_file_name_tl
      {
```

```
        \exp_after:wN \__primargs_get_file_name_char_ii:w
        \token_to_meaning:N \g_primargs_token
        \q_stop
      }
    \primargs_read_x_token:N \__primargs_get_file_name_test:
  }
\cs_new:Npn \__primargs_get_file_name_char_ii:w #1 ~ #2 ~ #3 \q_stop
  { \__primargs_get_file_name_char_iii:w #3 ~ #3 ~ \q_stop }
\cs_new:Npn \__primargs_get_file_name_char_iii:w #1 ~ #2 ~ #3 \q_stop {#2}
```

(*End definition for* `\__primargs_get_file_name_char:`, `\__primargs_get_file_name_char_ii:w`, *and* `\__primargs_get_file_name_char_iii:w`.)

`\primargs_get_input_file_name:N`
`\__primargs_get_input_file_name_first:`
`\__primargs_get_input_file_name_loop:`
`\__primargs_get_input_file_name_test:`
`\__primargs_get_input_file_name_brace:`
`\__primargs_get_input_file_name_aux:N`

For most engines this is an alias of `\primargs_get_file_name:N`. In LuaTEX we test for a catcode 1 token (after a filler) then expand and collect tokens (turned to strings) one by one, counting begin-group and end-group tokens in `\g__primargs_file_name_level_tl`. The control sequence `\par` is ignored. After removing a filler or after expansion, `\g_primargs_token` cannot be `\outer` hence the tests are safe. We use primitives to cope with outer macro hidden by `\noexpand` upon first expansion.

```
\sys_if_engine_luatex:TF
  {
    \cs_new_protected:Npn \primargs_get_input_file_name:N #1
      {
        \group_begin:
          \__primargs_safe:
          \tex_aftergroup:D #1
          \tl_gclear:N \g__primargs_file_name_tl
          \tl_gset:Nn \g__primargs_file_name_level_tl { 1 }
          \primargs_remove_filler:N \__primargs_get_input_file_name_first:
      }
    \cs_new_protected:Npn \__primargs_get_input_file_name_first:
      {
        \token_if_eq_catcode:NNTF \g_primargs_token \c_group_begin_token
          { \primargs_remove_token:N \__primargs_get_input_file_name_loop: }
          { \primargs_get_file_name:N \group_end: }
      }
    \cs_new_protected:Npn \__primargs_get_input_file_name_loop:
      { \primargs_read_x_token:N \__primargs_get_input_file_name_test: }
    \cs_new_protected:Npn \__primargs_get_input_file_name_test:
      {
        \token_if_eq_catcode:NNTF \g_primargs_token \c_group_begin_token
          {
            \tl_gset:Nx \g__primargs_file_name_level_tl
              { \int_eval:n { \g__primargs_file_name_level_tl + 1 } }
            \primargs_remove_token:N \__primargs_get_input_file_name_brace:
          }
          {
            \token_if_eq_catcode:NNTF \g_primargs_token \c_group_end_token
              {
                \tl_gset:Nx \g__primargs_file_name_level_tl
                  { \int_eval:n { \g__primargs_file_name_level_tl - 1 } }
                \int_compare:nNnTF { \g__primargs_file_name_level_tl } > 0
                  { \primargs_remove_token:N \__primargs_get_input_file_name_brace: }
                  { \primargs_remove_token:N \__primargs_get_file_name_end: }
```

```
            }
            {
              \token_if_eq_meaning:NNTF \g_primargs_token \c_space_token
                {
                  \tl_gput_right:Nn \g__primargs_file_name_tl { ~ }
                  \primargs_remove_token:N \__primargs_get_input_file_name_loop:
                }
                { \exp_after:wN \__primargs_get_input_file_name_aux:N \exp_not:N }
            }
        }
    }
  \cs_new_protected:Npn \__primargs_get_input_file_name_brace:
    {
      \tl_gput_right:Nx \g__primargs_file_name_tl
        {
          \exp_after:wN \__primargs_get_file_name_char_ii:w
          \token_to_meaning:N \g_primargs_token
          \q_stop
        }
      \__primargs_get_input_file_name_loop:
    }
  \cs_new_protected:Npn \__primargs_get_input_file_name_aux:N #1
    {
      \exp_after:wN \str_if_eq_x:nnT
      \exp_after:wN { \token_to_str:N #1 } { \token_to_str:N \par }
        { \use_none:nnn }
      \tex_xdef:D \g__primargs_file_name_tl
        {
          \g__primargs_file_name_tl
          \exp_after:wN \tl_to_str:n \exp_after:wN { \exp_not:N #1 }
        }
      \__primargs_get_input_file_name_loop:
    }
}
{ \cs_new_eq:NN \primargs_get_input_file_name:N \primargs_get_file_name:N }
```

(*End definition for* `\primargs_get_input_file_name:N` *and others. These functions are documented on page 3.*)

</package>

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols

C

cs commands:

Q

R

S

T

U