

# l3file 模块

## 文件与 I/O 操作

LaTeX 项目组\* 2024 年 1 月 04 日 发布

张泓知 2024 年 1 月 19 日 【译】

### 目 录

<b>1</b>	<b>Input-output stream management</b>	<b>3</b>
1.1	Reading from files	5
1.2	Reading from the terminal	9
1.3	Writing to files	10
1.4	Wrapping lines in output	13
1.5	Constant input-output streams, and variables	14
1.6	Primitive conditionals	15
<b>2</b>	<b>File operations</b>	<b>15</b>
2.1	Basic file operations	15
2.2	Information about files and file contents	16
2.3	Accessing file contents	20
<b>3</b>	<b>l3file implementation</b>	<b>21</b>
3.1	Input operations	21
3.1.1	Variables and constants	21
3.1.2	Stream management	22
3.1.3	Reading input	27
3.2	Output operations	30
3.2.1	Variables and constants	30
3.2.2	Internal auxiliaries	32

---

\*E-mail: latex-team@latex-project.org

3.3	Stream management . . . . .	32
3.3.1	Deferred writing . . . . .	35
3.3.2	Immediate writing . . . . .	36
3.3.3	Special characters for writing . . . . .	37
3.3.4	Hard-wrapping lines to a character count . . . . .	37
3.4	File operations . . . . .	49
3.4.1	Internal auxiliaries . . . . .	51
3.5	GetIdInfo . . . . .	71
3.6	Checking the version of kernel dependencies . . . . .	72
3.7	Messages . . . . .	75
3.8	Functions delayed from earlier modules . . . . .	76
<b>索引</b>	. . . . .	<b>77</b>

This module provides functions for working with external files. Some of these functions apply to an entire file, and have prefix `\file_...`, while others are used to work with files on a line by line basis and have prefix `\ior_...` (reading) or `\iow_...` (writing).

It is important to remember that when reading external files TeX attempts to locate them using both the operating system path and entries in the TeX file database (most TeX systems use such a database). Thus the “current path” for TeX is somewhat broader than that for other programs.

For functions which expect a *<file name>* argument, this argument may contain both literal items and expandable content, which should on full expansion be the desired file name. Active characters (as declared in `\l_char_active_seq`) are *not* expanded, allowing the direct use of these in file names. Quote tokens (") are not permitted in file names as they are reserved for internal use by some TeX primitives.

Spaces are trimmed at the beginning and end of the file name: this reflects the fact that some file systems do not allow or interact unpredictably with spaces in these positions. When no extension is given, this will trim spaces from the start of the name only.

## 1 Input–output stream management

As TeX engines have a limited number of input and output streams, direct use of the streams by the programmer is not supported in L<sup>A</sup>T<sub>E</sub>X3. Instead, an internal pool of streams is maintained, and these are allocated and deallocated as needed by other modules. As a result, the programmer should close streams when they are no longer needed, to release them for other processes.

Note that I/O operations are global: streams should all be declared with global names and treated accordingly.

---

<code>\ior_new:N</code>	<code>\ior_new:N &lt;stream&gt;</code>
<code>\ior_new:c</code>	<code>\iow_new:N &lt;stream&gt;</code>
<code>\iow_new:N</code>	Globally reserves the name of the <i>&lt;stream&gt;</i> , either for reading or for writing as appropriate. The <i>&lt;stream&gt;</i> is not opened until the appropriate <code>\..._open:Nn</code> function is used. Attempting to use a <i>&lt;stream&gt;</i> which has not been opened is an error, and the
<code>\iow_new:c</code>	<i>&lt;stream&gt;</i> will behave as the corresponding <code>\c_term_...</code>

---

New: 2011-09-26  
Updated: 2011-12-27

---

`\ior_open:Nn` `\ior_open:Nn <stream> {<file name>}`

`\ior_open:cn` Opens `<file name>` for reading using `<stream>` as the control sequence for file access.

Updated: 2012-02-10 If the `<stream>` was already open it is closed before the new operation begins. The `<stream>` is available for access immediately and will remain allocated to `<file name>` until a `\ior_close:N` instruction is given or the `TEX` run ends. If the file is not found, an error is raised.

---

`\ior_open:NnTF` `\ior_open:NnTF <stream> {<file name>} {<true code>} {<false code>}`

`\ior_open:cnTF` Opens `<file name>` for reading using `<stream>` as the control sequence for file access.

New: 2013-01-12 If the `<stream>` was already open it is closed before the new operation begins. The `<stream>` is available for access immediately and will remain allocated to `<file name>` until a `\ior_close:N` instruction is given or the `TEX` run ends. The `<true code>` is then inserted into the input stream. If the file is not found, no error is raised and the `<false code>` is inserted into the input stream.

---

`\iow_open:Nn` `\iow_open:Nn <stream> {<file name>}`

`\iow_open:(NV|cn|cV)` Opens `<file name>` for writing using `<stream>` as the control sequence for file access.

Updated: 2012-02-09 If the `<stream>` was already open it is closed before the new operation begins. The `<stream>` is available for access immediately and will remain allocated to `<file name>` until a `\iow_close:N` instruction is given or the `TEX` run ends. Opening a file for writing clears any existing content in the file (*i.e.* writing is *not* additive).

---

`\ior_shell_open:Nn` `\ior_shell_open:Nn <stream> {<shell command>}`

New: 2019-05-08 Opens the *pseudo*-file created by the output of the `<shell command>` for reading using `<stream>` as the control sequence for access. If the `<stream>` was already open it is closed before the new operation begins. The `<stream>` is available for access immediately and will remain allocated to `<shell command>` until a `\ior_close:N` instruction is given or the `TEX` run ends. If piped system calls are disabled an error is raised.

For details of handling of the `<shell command>`, see `\sys_get_shell:nnNTF`.

---

`\iow_shell_open:Nn` `\iow_shell_open:Nn`  $\langle stream \rangle$   $\{\langle shell\ command \rangle\}$

---

New: 2023-05-25

Opens the *pseudo*-file created by the output of the  $\langle shell\ command \rangle$  for writing using  $\langle stream \rangle$  as the control sequence for access. If the  $\langle stream \rangle$  was already open it is closed before the new operation begins. The  $\langle stream \rangle$  is available for access immediately and will remain allocated to  $\langle shell\ command \rangle$  until a `\iow_close:N` instruction is given or the T<sub>E</sub>X run ends. If piped system calls are disabled an error is raised.

For details of handling of the  $\langle shell\ command \rangle$ , see `\sys_get_shell:nnNTF`.

---

`\ior_close:N` `\ior_close:N`  $\langle stream \rangle$

`\ior_close:c` `\iow_close:N`  $\langle stream \rangle$

`\iow_close:N` Closes the  $\langle stream \rangle$ . Streams should always be closed when they are finished with  
`\iow_close:c` as this ensures that they remain available to other programmers.

Updated: 2012-07-31

---

`\ior_show:N` `\ior_show:N`  $\langle stream \rangle$

`\ior_show:c` `\ior_log:N`  $\langle stream \rangle$

`\ior_log:N` `\iow_show:N`  $\langle stream \rangle$

`\ior_log:c` `\iow_log:N`  $\langle stream \rangle$

`\iow_show:N` Display (to the terminal or log file) the file name associated to the (read or write)

`\iow_show:c`  $\langle stream \rangle$ .

`\iow_log:N`

`\iow_log:c`

New: 2021-05-11

---

`\ior_show_list:` `\ior_show_list:`

`\ior_log_list:` `\ior_log_list:`

`\iow_show_list:` `\iow_show_list:`

`\iow_log_list:` `\iow_log_list:`

New: 2017-06-27

Display (to the terminal or log file) a list of the file names associated with each open (read or write) stream. This is intended for tracking down problems.

## 1.1 Reading from files

Reading from files and reading from the terminal are separate processes in `expl3`. The functions `\ior_get:NN` and `\ior_str_get:NN`, and their branching equivalents, are designed to work with *files*.

---

`\ior_get:NN` `\ior_get:NN`  $\langle stream \rangle$   $\langle token\ list\ variable \rangle$

`\ior_get:NNTF` `\ior_get:NNTF`  $\langle stream \rangle$   $\langle token\ list\ variable \rangle$   $\langle true\ code \rangle$   $\langle false\ code \rangle$

---

New: 2012-06-24

Updated: 2019-03-23

---

Function that reads one or more lines (until an equal number of left and right braces are found) from the file input  $\langle stream \rangle$  and stores the result locally in the  $\langle token\ list \rangle$  variable. The material read from the  $\langle stream \rangle$  is tokenized by  $\text{\TeX}$  according to the category codes and `\endlinechar` in force when the function is used. Assuming normal settings, any lines which do not end in a comment character `%` have the line ending converted to a space, so for example input

a b c

results in a token list `a b c`. Any blank line is converted to the token `\par`. Therefore, blank lines can be skipped by using a test such as

```
\ior_get:NN \l_my_stream \l_tmpa_tl
\tl_set:Nn \l_tmpb_tl { \par }
\tl_if_eq:NNF \l_tmpa_tl \l_tmpb_tl
...
```

Also notice that if multiple lines are read to match braces then the resulting token list can contain `\par` tokens. In the non-branching version, where the  $\langle stream \rangle$  is not open the  $\langle tl\ var \rangle$  is set to `\q_no_value`.

**$\text{\TeX}$ hackers note:** This protected macro is a wrapper around the  $\text{\TeX}$  primitive `\read`. Regardless of settings,  $\text{\TeX}$  replaces trailing space and tab characters (character codes 32 and 9) in each line by an end-of-line character (character code `\endlinechar`, omitted if `\endlinechar` is negative or too large) before turning characters into tokens according to current category codes. With default settings, spaces appearing at the beginning of lines are also ignored.

---

<code>\ior_str_get:NN</code>	<code>\ior_str_get:NN &lt;stream&gt; &lt;token list variable&gt;</code>
<code>\ior_str_get:NNTF</code>	<code>\ior_str_get:NNTF &lt;stream&gt; &lt;token list variable&gt; &lt;true code&gt; &lt;false code&gt;</code>

---

New: 2016-12-04

Updated: 2019-03-23

Function that reads one line from the file input  $\langle stream \rangle$  and stores the result locally in the  $\langle token list \rangle$  variable. The material is read from the  $\langle stream \rangle$  as a series of tokens with category code 12 (other), with the exception of space characters which are given category code 10 (space). Multiple whitespace characters are retained by this process. It always only reads one line and any blank lines in the input result in the  $\langle token list variable \rangle$  being empty. Unlike `\ior_get:NN`, line ends do not receive any special treatment. Thus input

a b c

results in a token list `a b c` with the letters `a`, `b`, and `c` having category code 12. In the non-branching version, where the  $\langle stream \rangle$  is not open the  $\langle tl var \rangle$  is set to `\q_no_value`.

**T<sub>E</sub>Xhackers note:** This protected macro is a wrapper around the  $\varepsilon$ -T<sub>E</sub>X primitive `\readline`. Regardless of settings, T<sub>E</sub>X removes trailing space and tab characters (character codes 32 and 9). However, the end-line character normally added by this primitive is not included in the result of `\ior_str_get:NN`.

All mappings are done at the current group level, *i.e.* any local assignments made by the  $\langle function \rangle$  or  $\langle code \rangle$  discussed below remain in effect after the loop.

---

<code>\ior_map_inline:Nn</code>	<code>\ior_map_inline:Nn &lt;stream&gt; {\langle inline function \rangle}</code>
---------------------------------	--

---

New: 2012-02-11

Applies the  $\langle inline function \rangle$  to each set of  $\langle lines \rangle$  obtained by calling `\ior_get:NN` until reaching the end of the file. T<sub>E</sub>X ignores any trailing new-line marker from the file it reads. The  $\langle inline function \rangle$  should consist of code which receives the  $\langle line \rangle$  as #1.

---

<code>\ior_str_map_inline:Nn</code>	<code>\ior_str_map_inline:Nn &lt;stream&gt; {\langle inline function \rangle}</code>
-------------------------------------	--

---

New: 2012-02-11

Applies the  $\langle inline function \rangle$  to every  $\langle line \rangle$  in the  $\langle stream \rangle$ . The material is read from the  $\langle stream \rangle$  as a series of tokens with category code 12 (other), with the exception of space characters which are given category code 10 (space). The  $\langle inline function \rangle$  should consist of code which receives the  $\langle line \rangle$  as #1. Note that T<sub>E</sub>X removes trailing space and tab characters (character codes 32 and 9) from every line upon input. T<sub>E</sub>X also ignores any trailing new-line marker from the file it reads.

---

`\ior_map_variable:NNn` `\ior_map_variable:NNn`  $\langle stream \rangle$   $\langle tl\ var \rangle$   $\{\langle code \rangle\}$

---

New: 2019-01-13

For each set of  $\langle lines \rangle$  obtained by calling `\ior_get:NN` until reaching the end of the file, stores the  $\langle lines \rangle$  in the  $\langle tl\ var \rangle$  then applies the  $\langle code \rangle$ . The  $\langle code \rangle$  will usually make use of the  $\langle variable \rangle$ , but this is not enforced. The assignments to the  $\langle variable \rangle$  are local. Its value after the loop is the last set of  $\langle lines \rangle$ , or its original value if the  $\langle stream \rangle$  is empty.  $\text{\TeX}$  ignores any trailing new-line marker from the file it reads. This function is typically faster than `\ior_map_inline:Nn`.

---

`\ior_str_map_variable:NNn` `\ior_str_map_variable:NNn`  $\langle stream \rangle$   $\langle variable \rangle$   $\{\langle code \rangle\}$

---

New: 2019-01-13

For each  $\langle line \rangle$  in the  $\langle stream \rangle$ , stores the  $\langle line \rangle$  in the  $\langle variable \rangle$  then applies the  $\langle code \rangle$ . The material is read from the  $\langle stream \rangle$  as a series of tokens with category code 12 (other), with the exception of space characters which are given category code 10 (space). The  $\langle code \rangle$  will usually make use of the  $\langle variable \rangle$ , but this is not enforced. The assignments to the  $\langle variable \rangle$  are local. Its value after the loop is the last  $\langle line \rangle$ , or its original value if the  $\langle stream \rangle$  is empty. Note that  $\text{\TeX}$  removes trailing space and tab characters (character codes 32 and 9) from every line upon input.  $\text{\TeX}$  also ignores any trailing new-line marker from the file it reads. This function is typically faster than `\ior_str_map_inline:Nn`.

---

`\ior_map_break:` `\ior_map_break:`

---

New: 2012-06-29

Used to terminate a `\ior_map_...` function before all lines from the  $\langle stream \rangle$  have been processed. This normally takes place within a conditional statement, for example

```
\ior_map_inline:Nn \l_my_ior
{
  \str_if_eq:nnTF { #1 } { bingo }
  { \ior_map_break: }
  {
    % Do something useful
  }
}
```

Use outside of a `\ior_map_...` scenario leads to low level  $\text{\TeX}$  errors.

**$\text{\TeX}$ hackers note:** When the mapping is broken, additional tokens may be inserted before further items are taken from the input stream. This depends on the design of the mapping function.



---

`\ior_map_break:n` `\ior_map_break:n {<code>}`

---

New: 2012-06-29

Used to terminate a `\ior_map_...` function before all lines in the `<stream>` have been processed, inserting the `<code>` after the mapping has ended. This normally takes place within a conditional statement, for example

```
\ior_map_inline:Nn \l_my_ior
{
  \str_if_eq:nnTF { #1 } { bingo }
  { \ior_map_break:n { <code> } }
  {
    % Do something useful
  }
}
```

Use outside of a `\ior_map_...` scenario leads to low level T<sub>E</sub>X errors.

**T<sub>E</sub>Xhackers note:** When the mapping is broken, additional tokens may be inserted before the `<code>` is inserted into the input stream. This depends on the design of the mapping function.

---

`\ior_if_eof_p:N` `\ior_if_eof_p:N <stream>`

`\ior_if_eof:N $\underline{T$ F` `\ior_if_eof:N $\underline{T$ F` `<stream>` `{<true code>}` `{<false code>}`

---

Updated: 2012-02-10

Tests if the end of a file `<stream>` has been reached during a reading operation. The test also returns a `true` value if the `<stream>` is not open.

## 1.2 Reading from the terminal

---

`\ior_get_term:nN` `\ior_get_term:nN <prompt> <token list variable>`

`\ior_str_get_term:nN`

---

New: 2019-03-23

Function that reads one or more lines (until an equal number of left and right braces are found) from the terminal and stores the result locally in the `<token list>` variable. Tokenization occurs as described for `\ior_get:N` or `\ior_str_get:N`, respectively. When the `<prompt>` is empty, T<sub>E</sub>X will wait for input without any other indication: typically the programmer will have provided a suitable text using e.g. `\iow_term:n`. Where the `<prompt>` is given, it will appear in the terminal followed by an `=`, e.g.

```
prompt=
```

### 1.3 Writing to files

---

<code>\iow_now:Nn</code>	<code>\iow_now:Nn &lt;stream&gt; {&lt;tokens&gt;}</code>
<code>\iow_now:(NV Ne cn cV ce)</code>	This function writes $\langle tokens \rangle$ to the specified $\langle stream \rangle$ immediately ( <i>i.e.</i> the write operation is called on expansion of <code>\iow_now:Nn</code> ).

---

Updated: 2012-06-05

---



---

<code>\iow_log:n</code>	<code>\iow_log:n {&lt;tokens&gt;}</code>
<code>\iow_log:e</code>	This function writes the given $\langle tokens \rangle$ to the log (transcript) file immediately: it is a dedicated version of <code>\iow_now:Nn</code> .

---



---

<code>\iow_term:n</code>	<code>\iow_term:n {&lt;tokens&gt;}</code>
<code>\iow_term:e</code>	This function writes the given $\langle tokens \rangle$ to the terminal file immediately: it is a dedicated version of <code>\iow_now:Nn</code> .

---



---

<code>\iow_shipout:Nn</code>	<code>\iow_shipout:Nn &lt;stream&gt; {&lt;tokens&gt;}</code>
<code>\iow_shipout:(Ne cn ce)</code>	This function writes $\langle tokens \rangle$ to the specified $\langle stream \rangle$ when the current page is finalised ( <i>i.e.</i> at shipout). The <b>e</b> -type variants expand the $\langle tokens \rangle$ at the point where the function is used but <i>not</i> when the resulting tokens are written to the $\langle stream \rangle$ ( <i>cf.</i> <code>\iow_shipout_e:Nn</code> ).

---

**T<sub>E</sub>Xhackers note:** When using `expl3` with a format other than L<sup>A</sup>T<sub>E</sub>X, new line characters inserted using `\iow_newline:` or using the line-wrapping code `\iow_wrap:nnnN` are not recognized in the argument of `\iow_shipout:Nn`. This may lead to the insertion of additional unwanted line-breaks.

---

<code>\iow_shipout_e:Nn</code>	<code>\iow_shipout_e:Nn &lt;stream&gt; {&lt;tokens&gt;}</code>
<code>\iow_shipout_e:(Ne cn ce)</code>	This function writes $\langle tokens \rangle$ to the specified $\langle stream \rangle$ when the current page is finalised ( <i>i.e.</i> at shipout). The $\langle tokens \rangle$ are expanded at the time of writing in addition to any expansion when the function is used. This makes these functions suitable for including material finalised during the page building process (such as the page number integer).

---

Updated: 2023-09-17

---

**T<sub>E</sub>Xhackers note:** This is a wrapper around the T<sub>E</sub>X primitive `\write`. When using `expl3` with a format other than L<sup>A</sup>T<sub>E</sub>X, new line characters inserted using `\iow_newline:` or using the line-wrapping code `\iow_wrap:nnnN` are not recognized in the argument of `\iow_shipout:Nn`. This may lead to the insertion of additional unwanted line-breaks.

---

`\iow_char:N` ★ `\iow_char:N \<char>`

---

Inserts  $\langle char \rangle$  into the output stream. Useful when trying to write difficult characters such as %, {, }, *etc.* in messages, for example:

```
\iow_now:Nn \g_my_iow { \iow_char:N \{ text \iow_char:N \} }
```

The function has no effect if writing is taking place without expansion (*e.g.* in the second argument of `\iow_now:Nn`).

---

`\iow_newline:` ★ `\iow_newline:`

---

Function to add a new line within the  $\langle tokens \rangle$  written to a file. The function has no effect if writing is taking place without expansion (*e.g.* in the second argument of `\iow_now:Nn`).

**TeXhackers note:** When using `expl3` with a format other than  $\text{\LaTeX}$ , the character inserted by `\iow_newline:` is not recognized by  $\text{\TeX}$ , which may lead to the insertion of additional unwanted line-breaks. This issue only affects `\iow_shipout:Nn`, `\iow_shipout_e:Nn` and direct uses of primitive operations.



## 1.4 Wrapping lines in output

---

`\iow_wrap:nnnN` `\iow_wrap:nnnN`  $\{\langle text \rangle\}$   $\{\langle run-on text \rangle\}$   $\{\langle set up \rangle\}$   $\langle function \rangle$

`\iow_wrap:nenN`

New: 2012-06-28

Updated: 2017-12-04

---

This function wraps the  $\langle text \rangle$  to a fixed number of characters per line. At the start of each line which is wrapped, the  $\langle run-on text \rangle$  is inserted. The line character count targeted is the value of `\l_iow_line_count_int` minus the number of characters in the  $\langle run-on text \rangle$  for all lines except the first, for which the target number of characters is simply `\l_iow_line_count_int` since there is no run-on text. The  $\langle text \rangle$  and  $\langle run-on text \rangle$  are exhaustively expanded by the function, with the following substitutions:

- `\` or `\iow_newline`: may be used to force a new line,
- `\_` may be used to represent a forced space (for example after a control sequence),
- `\#`, `\%`, `\{`, `\}`, `\~` may be used to represent the corresponding character,
- `\iow_wrap_allow_break`: may be used to allow a line-break without inserting a space,
- `\iow_indent:n` may be used to indent a part of the  $\langle text \rangle$  (not the  $\langle run-on text \rangle$ ).

Additional functions may be added to the wrapping by using the  $\langle set up \rangle$ , which is executed before the wrapping takes place: this may include overriding the substitutions listed.

Any expandable material in the  $\langle text \rangle$  which is not to be expanded on wrapping should be converted to a string using `\token_to_str:N`, `\tl_to_str:n`, `\tl_to_str:N`, *etc.*

The result of the wrapping operation is passed as a braced argument to the  $\langle function \rangle$ , which is typically a wrapper around a write operation. The output of `\iow_wrap:nnnN` (*i.e.* the argument passed to the  $\langle function \rangle$ ) consists of characters of category “other” (category code 12), with the exception of spaces which have category “space” (category code 10). This means that the output does *not* expand further when written to a file.

**T<sub>E</sub>Xhackers note:** Internally, `\iow_wrap:nnnN` carries out an e-type expansion on the  $\langle text \rangle$  to expand it. This is done in such a way that `\exp_not:N` or `\exp_not:n` *could* be used to prevent expansion of material. However, this is less conceptually clear than conversion to a string, which is therefore the supported method for handling expandable material in the  $\langle text \rangle$ .

---

`\iow_wrap_allow_break:` `\iow_wrap_allow_break:`

---

New: 2023-04-25

In the first argument of `\iow_wrap:nnnN` (for instance in messages), inserts a break-point that allows a line break. If no break occurs, this function adds nothing to the output.

---

`\iow_indent:n` `\iow_indent:n {\text}`

---

New: 2011-09-21

In the first argument of `\iow_wrap:nnnN` (for instance in messages), indents `\text` by four spaces. This function does not cause a line break, and only affects lines which start within the scope of the `\text`. In case the indented `\text` should appear on separate lines from the surrounding text, use `\\` to force line breaks.

---

`\l_iow_line_count_int`

---

New: 2012-06-24

The maximum number of characters in a line to be written by the `\iow_wrap:nnnN` function. This value depends on the  $\text{\TeX}$  system in use: the standard value is 78, which is typically correct for unmodified  $\text{\TeX}$  Live and  $\text{\MiKTeX}$  systems.

## 1.5 Constant input–output streams, and variables

---

`\g_tmpa_iow`

---

`\g_tmpb_iow`

---

New: 2017-12-11

Scratch input stream for global use. These are never used by the kernel code, and so are safe for use with any  $\text{\LaTeX}$ 3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

---

`\c_log_iow`

---

`\c_term_iow`

---

Constant output streams for writing to the log and to the terminal (plus the log), respectively.

---

`\g_tmpa_iow`

---

`\g_tmpb_iow`

---

New: 2017-12-11

Scratch output stream for global use. These are never used by the kernel code, and so are safe for use with any  $\text{\LaTeX}$ 3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

## 1.6 Primitive conditionals

---

`\if_eof:w` ★ `\if_eof:w`  $\langle stream \rangle$   
     $\langle true\ code \rangle$   
`\else:`  
     $\langle false\ code \rangle$   
`\fi:`

Tests if the  $\langle stream \rangle$  returns “end of file”, which is true for non-existent files. The `\else:` branch is optional.

**TeXhackers note:** This is the TeX primitive `\ifeof`.

## 2 File operations

### 2.1 Basic file operations

---

`\g_file_curr_dir_str`  
`\g_file_curr_name_str`  
`\g_file_curr_ext_str`

New: 2017-06-21

Contain the directory, name and extension of the current file. The directory is empty if the file was loaded without an explicit path (*i.e.* if it is in the TeX search path), and does *not* end in / other than the case that it is exactly equal to the root directory. The  $\langle name \rangle$  and  $\langle ext \rangle$  parts together make up the file name, thus the  $\langle name \rangle$  part may be thought of as the “job name” for the current file.

Note that TeX does not provide information on the  $\langle dir \rangle$  and  $\langle ext \rangle$  part for the main (top level) file and that this file always has empty  $\langle dir \rangle$  and  $\langle ext \rangle$  components. Also, the  $\langle name \rangle$  here will be equal to `\c_sys_jobname_str`, which may be different from the real file name (if set using `--jobname`, for example).

---

`\l_file_search_path_seq`

New: 2017-06-18

Updated: 2023-06-15

Each entry is the path to a directory which should be searched when seeking a file. Each path can be relative or absolute, and need not include the trailing slash. Spaces need not be quoted.

**TeXhackers note:** When working as a package in L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>, `expl3` will automatically append the current `\input@path` to the set of values from `\l_file_search_path_seq`.

---

<code>\file_if_exist_p:n</code>	★	<code>\file_if_exist_p:n {⟨file name⟩}</code>
<code>\file_if_exist_p:V</code>	★	<code>\file_if_exist:nTF {⟨file name⟩} {⟨true code⟩} {⟨false code⟩}</code>
<code>\file_if_exist:nTF</code>	★	Expands the argument of the <code>\file name</code> to give a string, then searches for this
<code>\file_if_exist:VTF</code>	★	string using the current T <sub>E</sub> X search path and the additional paths controlled by
Updated: 2023-09-18		<code>\l_file_search_path_seq.</code>

---

## 2.2 Information about files and file contents

Functions in this section return information about files as expl3 `str` data, *except* that the non-expandable functions set their return *token list* to `\q_no_value` if the file requested is not found. As such, comparison of file names, hashes, sizes, etc., should use `\str_if_eq:nnTF` rather than `\tl_if_eq:nnTF` and so on.

---

<code>\file_hex_dump:n</code>	☆	<code>\file_hex_dump:n {⟨file name⟩}</code>
<code>\file_hex_dump:V</code>	☆	<code>\file_hex_dump:nnn {⟨file name⟩} {⟨start index⟩} {⟨end index⟩}</code>
<code>\file_hex_dump:nnn</code>	☆	Searches for <code>⟨file name⟩</code> using the current T <sub>E</sub> X search path and the additional paths
<code>\file_hex_dump:Vnn</code>	☆	controlled by <code>\l_file_search_path_seq</code> . It then expands to leave the hexadecimal
New: 2019-11-19		dump of the file content in the input stream. The file is read as bytes, which means
		that in contrast to most T <sub>E</sub> X behaviour there will be a difference in result depending
		on the line endings used in text files. The same file will produce the same result
		between different engines: the algorithm used is the same in all cases. When the
		file is not found, the result of expansion is empty. The <code>{⟨start index⟩}</code> and <code>{⟨end</code>
		<code>index⟩}</code> values work as described for <code>\str_range:nnn</code> .

---



---

<code>\file_get_hex_dump:nN</code>	<code>\file_get_hex_dump:nN {⟨file name⟩} ⟨tl var⟩</code>
<code>\file_get_hex_dump:VN</code>	<code>\file_get_hex_dump:nnnN {⟨file name⟩} {⟨start index⟩} {⟨end index⟩} ⟨tl var⟩</code>
<code>\file_get_hex_dump:nNTF</code>	Sets the <code>⟨tl var⟩</code> to the result of applying <code>\file_hex_dump:n/\file_hex_dump:nnn</code>
<code>\file_get_hex_dump:VNTF</code>	to the <code>⟨file⟩</code> . If the file is not found, the <code>⟨tl var⟩</code> will be set to <code>\q_no_value</code> .
<code>\file_get_hex_dump:nnnN</code>	
<code>\file_get_hex_dump:VnnN</code>	
<code>\file_get_hex_dump:nnnNTF</code>	
<code>\file_get_hex_dump:VnnNTF</code>	
New: 2019-11-19	

---



---

`\file_md5hash:n` ☆ `\file_md5hash:n {<file name>}`

`\file_md5hash:V` ☆

---

New: 2019-09-03

Searches for `<file name>` using the current  $\TeX$  search path and the additional paths controlled by `\l_file_search_path_seq`. It then expands to leave the MD5 sum generated from the contents of the file in the input stream. The file is read as bytes, which means that in contrast to most  $\TeX$  behaviour there will be a difference in result depending on the line endings used in text files. The same file will produce the same result between different engines: the algorithm used is the same in all cases. When the file is not found, the result of expansion is empty.

---

`\file_get_md5hash:nN` `\file_get_md5hash:nN {<file name>} <tl var>`

`\file_get_md5hash:VN`

`\file_get_md5hash:nN $\overline{TF}$`

`\file_get_md5hash:VN $\overline{TF}$`

---

New: 2017-07-11

Updated: 2019-02-16

Sets the `<tl var>` to the result of applying `\file_md5hash:n` to the `<file>`. If the file is not found, the `<tl var>` will be set to `\q_no_value`.

---

`\file_size:n` ☆ `\file_size:n {<file name>}`

`\file_size:V` ☆

---

New: 2019-09-03

Searches for `<file name>` using the current  $\TeX$  search path and the additional paths controlled by `\l_file_search_path_seq`. It then expands to leave the size of the file in bytes in the input stream. When the file is not found, the result of expansion is empty.

---

`\file_get_size:nN` `\file_get_size:nN {<file name>} <tl var>`

`\file_get_size:VN`

`\file_get_size:nN $\overline{TF}$`

`\file_get_size:VN $\overline{TF}$`

---

New: 2017-07-09

Updated: 2019-02-16

Sets the `<tl var>` to the result of applying `\file_size:n` to the `<file>`. If the file is not found, the `<tl var>` will be set to `\q_no_value`. This is not available in older versions of  $\XeTeX$ .

---

`\file_timestamp:n` ☆ `\file_timestamp:n {<file name>}`

`\file_timestamp:V` ☆

---

New: 2019-09-03

Searches for `<file name>` using the current  $\TeX$  search path and the additional paths controlled by `\l_file_search_path_seq`. It then expands to leave the modification timestamp of the file in the input stream. The timestamp is of the form `D:<year><month><day><hour><minute><second><offset>`, where the latter may be `Z` (UTC) or `<plus-minus><hours>'<minutes>'`. When the file is not found, the result of expansion is empty. This is not available in older versions of  $\XeTeX$ .

---

<code>\file_get_timestamp:nN</code>	<code>\file_get_timestamp:nN {&lt;file name&gt;} &lt;tl var&gt;</code>
<code>\file_get_timestamp:VN</code>	Sets the <code>&lt;tl var&gt;</code> to the result of applying <code>\file_timestamp:n</code> to the <code>&lt;file&gt;</code> . If the
<code>\file_get_timestamp:nN<math>\overline{TF}</math></code>	file is not found, the <code>&lt;tl var&gt;</code> will be set to <code>\q_no_value</code> . This is not available in
<code>\file_get_timestamp:VN<math>\overline{TF}</math></code>	older versions of X <sub>Y</sub> TeX.

---

New: 2017-07-09

Updated: 2019-02-16

---



---

<code>\file_compare_timestamp_p:nNn</code>	<code>★ \file_compare_timestamp_p:nNn {&lt;file-1&gt;} &lt;comparator&gt;</code>
<code>\file_compare_timestamp_p:(nNV VNn VNV)</code>	<code>★ {&lt;file-2&gt;}</code>
<code>\file_compare_timestamp:nNn<math>\overline{TF}</math></code>	<code>★ \file_compare_timestamp:nNn<math>\overline{TF}</math> {&lt;file-1&gt;} &lt;comparator&gt;</code>
<code>\file_compare_timestamp:(nNV VNn VNV)<math>\overline{TF}</math></code>	<code>★ {&lt;file-2&gt;} {&lt;true code&gt;} {&lt;false code&gt;}</code>

---

New: 2019-05-13

Updated: 2019-09-20

---

Compares the file stamps on the two `<files>` as indicated by the `<comparator>`, and inserts either the `<true code>` or `<false case>` as required. A file which is not found is treated as older than any file which is found. This allows for example the construct

```
\file_compare_timestamp:nNnT { source-file } > { derived-file }
{
  % Code to regenerate derived file
}
```

to work when the derived file is entirely absent. The timestamp of two absent files is regarded as different. This is not available in older versions of X<sub>Y</sub>TeX.

---

<code>\file_get_full_name:nN</code>	<code>\file_get_full_name:nN {&lt;file name&gt;} &lt;tl&gt;</code>
<code>\file_get_full_name:VN</code>	<code>\file_get_full_name:nNTF {&lt;file name&gt;} &lt;tl&gt; {&lt;true code&gt;} {&lt;false code&gt;}</code>
<code>\file_get_full_name:nN<math>\overline{TF}</math></code>	Searches for <code>&lt;file name&gt;</code> in the path as detailed for <code>\file_if_exist:nTF</code> , and if
<code>\file_get_full_name:VN<math>\overline{TF}</math></code>	found sets the <code>&lt;tl var&gt;</code> the fully-qualified name of the file, <i>i.e.</i> the path and file name.

---

Updated: 2019-02-16

---

This includes an extension `.tex` when the given `<file name>` has no extension but the file found has that extension. In the non-branching version, the `<tl var>` will be set to `\q_no_value` in the case that the file does not exist.

---

<code>\file_full_name:n ☆</code>	<code>\file_full_name:n {&lt;file name&gt;}</code>
<code>\file_full_name:V ☆</code>	Searches for <code>&lt;file name&gt;</code> in the path as detailed for <code>\file_if_exist:nTF</code> , and if
	found leaves the fully-qualified name of the file, <i>i.e.</i> the path and file name, in the
	input stream. This includes an extension <code>.tex</code> when the given <code>&lt;file name&gt;</code> has no
	extension but the file found has that extension. If the file is not found on the path,
	the expansion is empty.

---

New: 2019-09-03

---

`\file_parse_full_name:nNNN` `\file_parse_full_name:nNNN`  $\{\langle full\ name\rangle\}$   $\langle dir\rangle$   $\langle name\rangle$   $\langle ext\rangle$

`\file_parse_full_name:VNNN`

New: 2017-06-23

Updated: 2020-06-24

Parses the  $\langle full\ name\rangle$  and splits it into three parts, each of which is returned by setting the appropriate local string variable:

- The  $\langle dir\rangle$ : everything up to the last / (path separator) in the  $\langle file\ path\rangle$ . As with system PATH variables and related functions, the  $\langle dir\rangle$  does *not* include the trailing / unless it points to the root directory. If there is no path (only a file name),  $\langle dir\rangle$  is empty.
- The  $\langle name\rangle$ : everything after the last / up to the last ., where both of those characters are optional. The  $\langle name\rangle$  may contain multiple . characters. It is empty if  $\langle full\ name\rangle$  consists only of a directory name.
- The  $\langle ext\rangle$ : everything after the last . (including the dot). The  $\langle ext\rangle$  is empty if there is no . after the last /.

Before parsing, the  $\langle full\ name\rangle$  is expanded until only non-expandable tokens remain, except that active characters are also not expanded. Quotes (") are invalid in file names and are discarded from the input.

---

`\file_parse_full_name:n`  $\star$  `\file_parse_full_name:n`  $\{\langle full\ name\rangle\}$

`\file_parse_full_name:V`  $\star$  Parses the  $\langle full\ name\rangle$  as described for `\file_parse_full_name:nNNN`, and leaves  $\langle dir\rangle$ ,  $\langle name\rangle$ , and  $\langle ext\rangle$  in the input stream, each inside a pair of braces.

New: 2020-06-24

---

`\file_parse_full_name_apply:nN`  $\star$  `\file_parse_full_name_apply:nN`  $\{\langle full\ name\rangle\}$   $\langle function\rangle$

`\file_parse_full_name_apply:VN`  $\star$

New: 2020-06-24

Parses the  $\langle full\ name\rangle$  as described for `\file_parse_full_name:nNNN`, and passes  $\langle dir\rangle$ ,  $\langle name\rangle$ , and  $\langle ext\rangle$  as arguments to  $\langle function\rangle$ , as an n-type argument each, in this order.

## 2.3 Accessing file contents

---

<code>\file_get:nnN</code>	<code>\file_get:nnN {&lt;file name&gt;} {&lt;setup&gt;} &lt;tl&gt;</code>
<code>\file_get:VnN</code>	<code>\file_get:nnNTF {&lt;file name&gt;} {&lt;setup&gt;} &lt;tl&gt; {&lt;true code&gt;} {&lt;false code&gt;}</code>
<code>\file_get:nnNTF</code>	Defines <code>&lt;tl&gt;</code> to the contents of <code>&lt;file name&gt;</code> . Category codes may need to be set
<code>\file_get:VnNTF</code>	appropriately via the <code>&lt;setup&gt;</code> argument. The non-branching version sets the <code>&lt;tl&gt;</code> to
<small>New: 2019-01-16</small>	<code>\q_no_value</code> if the file is not found. The branching version runs the <code>&lt;true code&gt;</code>
<small>Updated: 2019-02-16</small>	after the assignment to <code>&lt;tl&gt;</code> if the file is found, and <code>&lt;false code&gt;</code> otherwise. The file
	content will be tokenized using the current category code régime,

---



---

<code>\file_input:n</code>	<code>\file_input:n {&lt;file name&gt;}</code>
<code>\file_input:V</code>	Searches for <code>&lt;file name&gt;</code> in the path as detailed for <code>\file_if_exist:nTF</code> , and if
<small>Updated: 2017-06-26</small>	found reads in the file as additional L <sup>A</sup> T <sub>E</sub> X source. All files read are recorded for
	information and the file name stack is updated by this function. An error is raised
	if the file is not found.

---



---

<code>\file_input_raw:n</code> *	<code>\file_input_raw:n {&lt;file name&gt;}</code>
<code>\file_input_raw:V</code> *	Searches for <code>&lt;file name&gt;</code> in the path as detailed for <code>\file_if_exist:nTF</code> , and if
<small>New: 2023-05-18</small>	found reads in the file as additional T <sub>E</sub> X source. No data concerning the file is
	tracked. If the file is not found, no action is taken.

---

**T<sub>E</sub>Xhackers note:** This function is intended only for contexts where files must be read purely by expansion, for example at the start of a table cell in an `\halign`.

---

<code>\file_if_exist_input:n</code>	<code>\file_if_exist_input:n {&lt;file name&gt;}</code>
<code>\file_if_exist_input:V</code>	<code>\file_if_exist_input:nF {&lt;file name&gt;} {&lt;false code&gt;}</code>
<code>\file_if_exist_input:nF</code>	Searches for <code>&lt;file name&gt;</code> using the current T <sub>E</sub> X search path and the additional paths
<code>\file_if_exist_input:VF</code>	included in <code>\l_file_search_path_seq</code> . If found then reads in the file as additional
<small>New: 2014-07-02</small>	L <sup>A</sup> T <sub>E</sub> X source as described for <code>\file_input:n</code> , otherwise inserts the <code>&lt;false code&gt;</code> .
	Note that these functions do not raise an error if the file is not found, in contrast to
	<code>\file_input:n</code> .

---

---

`\file_input_stop:` `\file_input_stop:`

---

New: 2017-07-07

Ends the reading of a file started by `\file_input:n` or similar before the end of the file is reached. Where the file reading is being terminated due to an error, `\msg_critical:nn(nn)` should be preferred.

**TeXhackers note:** This function must be used on a line on its own: TeX reads files line-by-line and so any additional tokens in the “current” line will still be read.

This is also true if the function is hidden inside another function (which will be the normal case), i.e., all tokens on the same line in the source file are still processed. Putting it on a line by itself in the definition doesn’t help as it is the line where it is used that counts!

---

`\file_show_list:` `\file_show_list:`

`\file_log_list:` `\file_log_list:`

---

These functions list all files loaded by L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> commands that populate `\@filelist` or by `\file_input:n`. While `\file_show_list:` displays the list in the terminal, `\file_log_list:` outputs it to the log file only.

## 3 l3file implementation

*The following test files are used for this code: m3file001.*

1 `<*package>`

### 3.1 Input operations

2 `<@@=ior>`

#### 3.1.1 Variables and constants

`\l__ior_internal_tl` Used as a short-term scratch variable.

3 `\tl_new:N \l__ior_internal_tl`

(`\l__ior_internal_tl` 定义结束。)

`\c__ior_term_ior` Reading from the terminal (with a prompt) is done using a positive but non-existent stream number. Unlike writing, there is no concept of reading from the log.

4 `\int_const:Nn \c__ior_term_ior { 16 }`

(`\c__ior_term_ior` 定义结束。)

`\g__ior_streams_seq` A list of the currently-available input streams to be used as a stack.

5 `\seq_new:N \g__ior_streams_seq`

(\g\_\_ior\_streams\_seq 定义结束。)

`\l__ior_stream_tl` Used to recover the raw stream number from the stack.

```
6 \tl_new:N \l__ior_stream_tl
```

(\l\_\_ior\_stream\_tl 定义结束。)

`\g__ior_streams_prop` The name of the file attached to each stream is tracked in a property list. To get the correct number of reserved streams in package mode the underlying mechanism needs to be queried. For L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> and plain T<sub>E</sub>X this data is stored in `\count16`: with the `etex` package loaded we need to subtract 1 as the register holds the number of the next stream to use. In ConT<sub>E</sub>Xt, we need to look at `\count38` but there is no subtraction: like the original plain T<sub>E</sub>X/L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> mechanism it holds the value of the *last* stream allocated.

```
7 \prop_new:N \g__ior_streams_prop
8 \int_step_inline:nnn
9 { 0 }
10 {
11   \cs_if_exist:NTF \contextversion
12     { \tex_count:D 38 ~ }
13     {
14       \tex_count:D 16 ~ %
15       \cs_if_exist:NT \loccount { - 1 }
16     }
17 }
18 {
19   \prop_gput:Nnn \g__ior_streams_prop {#1} { Reserved~by~format }
20 }
```

(\g\_\_ior\_streams\_prop 定义结束。)

### 3.1.2 Stream management

`\ior_new:N` Reserving a new stream is done by defining the name as equal to using the terminal.

```
\ior_new:c 21 \cs_new_protected:Npn \ior_new:N #1 { \cs_new_eq:NN #1 \c__ior_term_ior }
22 \cs_generate_variant:Nn \ior_new:N { c }
```

(\ior\_new:N 定义结束。这个函数被记录在第9页。)

`\g_tmpa_ior` The usual scratch space.

```
\g_tmpb_ior 23 \ior_new:N \g_tmpa_ior
24 \ior_new:N \g_tmpb_ior
```

(`\g_tmpa_ior` 和 `\g_tmpb_ior` 定义结束。这些变量被记录在第14页。)

`\ior_open:Nn` Use the conditional version, with an error if the file is not found.

```
\ior_open:cn 25 \cs_new_protected:Npn \ior_open:Nn #1#2
26 { \ior_open:NnF #1 {#2} { \__kernel_file_missing:n {#2} } }
27 \cs_generate_variant:Nn \ior_open:Nn { c }
```

(`\ior_open:Nn` 定义结束。这个函数被记录在第4页。)

`\l__ior_file_name_tl` Data storage.

```
28 \tl_new:N \l__ior_file_name_tl
```

(`\l__ior_file_name_tl` 定义结束。)

`\ior_open:NnTF` An auxiliary searches for the file in the  $\text{\TeX}$ ,  $\text{\LaTeX 2}_{\epsilon}$  and  $\text{\LaTeX 3}$  paths. Then pass the file found to the lower-level function which deals with streams. The `full_name` is empty when the file is not found.

```
\ior_open:cnTF 29 \prg_new_protected_conditional:Npnn \ior_open:Nn #1#2 { T , F , TF }
30 {
31   \file_get_full_name:nNTF {#2} \l__ior_file_name_tl
32   {
33     \__kernel_ior_open:No #1 \l__ior_file_name_tl
34     \prg_return_true:
35   }
36   { \prg_return_false: }
37 }
38 \prg_generate_conditional_variant:Nnn \ior_open:Nn { c } { T , F , TF }
```

(`\ior_open:NnTF` 定义结束。这个函数被记录在第4页。)

`\__ior_new:N` Streams are reserved using `\newread` before they can be managed by `ior`. To prevent `ior` from being affected by redefinitions of `\newread` (such as done by the third-party package `morewrites`), this macro is saved here under a private name. The complicated code ensures that `\__ior_new:N` is not `\outer` despite plain  $\text{\TeX}$ 's `\newread` being `\outer`. For  $\text{\ConTeXt}$ , we have to deal with the fact that `\newread` works like our own: it actually checks before altering definition.

```
39 \exp_args:NNf \cs_new_protected:Npn \__ior_new:N
40 { \exp_args:NNc \exp_after:wN \exp_stop_f: { newread } }
41 \cs_if_exist:NT \contextversion
42 {
43   \cs_new_eq:NN \__ior_new_aux:N \__ior_new:N
44   \cs_gset_protected:Npn \__ior_new:N #1
```

```

45     {
46         \cs_undefine:N #1
47         \__ior_new_aux:N #1
48     }
49 }

```

(`\__ior_new:N` 定义结束。)

`\__kernel_ior_open:Nn` The stream allocation itself uses the fact that there is a list of all of those available.  
`\__kernel_ior_open:No` Life gets more complex as it's important to keep things in sync. That is done using  
`\__ior_open_stream:Nn` a two-part approach: any streams that have already been taken up by `ior` but are now free are tracked, so we first try those. If that fails, ask plain  $\text{\TeX}$  or  $\text{\LaTeX} 2_{\epsilon}$  for a new stream and use that number (after a bit of conversion).

```

50 \cs_new_protected:Npn \__kernel_ior_open:Nn #1#2
51 {
52     \ior_close:N #1
53     \seq_gpop:NNTF \g__ior_streams_seq \l__ior_stream_tl
54     { \__ior_open_stream:Nn #1 {#2} }
55     {
56         \__ior_new:N #1
57         \__kernel_tl_set:Ne \l__ior_stream_tl { \int_eval:n {#1} }
58         \__ior_open_stream:Nn #1 {#2}
59     }
60 }
61 \cs_generate_variant:Nn \__kernel_ior_open:Nn { No }

```

Here, we act defensively in case  $\text{\LuaTeX}$  is in use with an extensionless file name.

```

62 \cs_new_protected:Npe \__ior_open_stream:Nn #1#2
63 {
64     \tex_global:D \tex_chardef:D #1 = \exp_not:N \l__ior_stream_tl \scan_stop:
65     \prop_gput:NVn \exp_not:N \g__ior_streams_prop #1 {#2}
66     \tex_openin:D #1
67     \sys_if_engine luatex:TF
68     { {#2} }
69     { \exp_not:N \__kernel_file_name_quote:n {#2} \scan_stop: }
70 }

```

(`\__kernel_ior_open:Nn` 和 `\__ior_open_stream:Nn` 定义结束。)

`\ior_shell_open:Nn` Actually much easier than either the standard `open` or `input` versions! When calling  
`\__ior_shell_open:nN` `\__kernel_ior_open:Nn` the file the pipe is added to signal a shell command, but the  
`\__ior_shell_open:oN` quotes are not added yet—they are added later by `\__kernel_file_name_quote:n`.



```

71 \cs_new_protected:Npn \ior_shell_open:Nn #1#2
72 {
73   \sys_if_shell:TF
74     { \__ior_shell_open:oN { \tl_to_str:n {#2} } #1 }
75     { \msg_error:nn { kernel } { pipe-failed } }
76 }
77 \cs_new_protected:Npn \__ior_shell_open:nN #1#2
78 {
79   \tl_if_in:nnTF {#1} { " }
80   {
81     \msg_error:nne
82       { kernel } { quote-in-shell } {#1}
83   }
84   { \__kernel_ior_open:Nn #2 { |#1 } }
85 }
86 \cs_generate_variant:Nn \__ior_shell_open:nN { o }
87 \msg_new:nnnn { kernel } { pipe-failed }
88 { Cannot~run~piped~system~commands. }
89 {
90   LaTeX~tried~to~call~a~system~process~but~this~was~not~possible.\\
91   Try~the~"--shell-escape"~(or~"--enable-pipes")~option.
92 }

```

(*\ior\_shell\_open:Nn* 和 *\\_\_ior\_shell\_open:nN* 定义结束。这个函数被记录在第4页。)

**\ior\_close:N** Closing a stream means getting rid of it at the  $\text{\TeX}$  level and removing from the various data structures. Unless the name passed is an invalid stream number (outside the range  $[0, 15]$ ), it can be closed. On the other hand, it only gets added to the stack if it was not already there, to avoid duplicates building up.

```

93 \cs_new_protected:Npn \ior_close:N #1
94 {
95   \int_compare:nT { -1 < #1 < \c__ior_term_ior }
96   {
97     \tex_closein:D #1
98     \prop_gremove:NV \g__ior_streams_prop #1
99     \seq_if_in:NVF \g__ior_streams_seq #1
100     { \seq_gpush:NV \g__ior_streams_seq #1 }
101     \cs_gset_eq:NN #1 \c__ior_term_ior
102   }
103 }
104 \cs_generate_variant:Nn \ior_close:N { c }

```

(*\ior\_close:N* 定义结束。这个函数被记录在第5页。)

```

\ior_show:N Seek the stream in the \g__ior_streams_prop list, then show the stream as open
\ior_log:N or closed accordingly.
\__ior_show:NN
105 \cs_new_protected:Npn \ior_show:N { \__ior_show:NN \tl_show:n }
106 \cs_generate_variant:Nn \ior_show:N { c }
107 \cs_new_protected:Npn \ior_log:N { \__ior_show:NN \tl_log:n }
108 \cs_generate_variant:Nn \ior_log:N { c }
109 \cs_new_protected:Npn \__ior_show:NN #1#2
110 {
111     \__kernel_chk_defined:NT #2
112     {
113         \prop_get:NVNTF \g__ior_streams_prop #2 \l__ior_internal_tl
114         {
115             \exp_args:Ne #1
116             { \token_to_str:N #2 ~ open: ~ \l__ior_internal_tl }
117         }
118         { \exp_args:Ne #1 { \token_to_str:N #2 ~ closed } }
119     }
120 }

```

(*\ior\_show:N*, *\ior\_log:N*, 和 *\\_\_ior\_show:NN* 定义结束。这些函数被记录在第5页。)

```

\ior_show_list: Show the property lists, but with some “pretty printing”. See the l3msg module.
\ior_log_list: The first argument of the message is ior (as opposed to iow) and the second is
\__ior_list:N empty if no read stream is open and non-empty (the list of streams formatted using
\msg_show_item_unbraced:nn) otherwise. The code of the message show-streams
takes care of translating ior/iow to English.

```

```

121 \cs_new_protected:Npn \ior_show_list: { \__ior_list:N \msg_show:nneeee }
122 \cs_new_protected:Npn \ior_log_list: { \__ior_list:N \msg_log:nneeee }
123 \cs_new_protected:Npn \__ior_list:N #1
124 {
125     #1 { kernel } { show-streams }
126     { ior }
127     {
128         \prop_map_function:NN \g__ior_streams_prop
129         \msg_show_item_unbraced:nn
130     }
131     { } { }
132 }

```

(*\ior\_show\_list:*, *\ior\_log\_list:*, 和 *\\_\_ior\_list:N* 定义结束。这些函数被记录在第5页。)

### 3.1.3 Reading input

`\if_eof:w` The primitive conditional

```
133 \cs_new_eq:NN \if_eof:w \tex_ifeof:D
```

(`\if_eof:w` 定义结束。这个函数被记录在第15页。)

`\ior_if_eof_p:N` To test if some particular input stream is exhausted the following conditional is provided. The primitive test can only deal with numbers in the range  $[0, 15]$  so we catch outliers (they are exhausted).

`\ior_if_eof:N $\underline{TF}$`

```
134 \prg_new_conditional:Npnn \ior_if_eof:N #1 { p , T , F , TF }
135 {
136   \if_int_compare:w -1 < #1
137     \if_int_compare:w #1 < \c__ior_term_ior
138       \if_eof:w #1
139         \prg_return_true:
140       \else:
141         \prg_return_false:
142       \fi:
143     \else:
144       \prg_return_true:
145     \fi:
146   \else:
147     \prg_return_true:
148   \fi:
149 }
```

(`\ior_if_eof:N $\underline{TF}$`  定义结束。这个函数被记录在第9页。)

`\ior_get:NN` And here we read from files.

`\__ior_get:NN`

`\ior_get:NN $\underline{TF}$`

```
150 \cs_new_protected:Npn \ior_get:NN #1#2
151 { \ior_get:NNF #1 #2 { \tl_set:Nn #2 { \q_no_value } } }
152 \cs_new_protected:Npn \__ior_get:NN #1#2
153 { \tex_read:D #1 to #2 }
154 \prg_new_protected_conditional:Npnn \ior_get:NN #1#2 { T , F , TF }
155 {
156   \ior_if_eof:N $\underline{TF}$  #1
157   { \prg_return_false: }
158   {
159     \__ior_get:NN #1 #2
160     \prg_return_true:
161   }
162 }
```

(`\ior_get:NN`, `\_ior_get:NN`, 和 `\ior_get:NNTF` 定义结束。这些函数被记录在第6页。)

`\ior_str_get:NN` Reading as strings is a more complicated wrapper, as we wish to remove the endline  
`\_ior_str_get:NN` character and restore it afterwards.

```
\ior_str_get:NNTF 163 \cs_new_protected:Npn \ior_str_get:NN #1#2
164 { \ior_str_get:NNTF #1 #2 { \tl_set:Nn #2 { \q_no_value } } }
165 \cs_new_protected:Npn \_ior_str_get:NN #1#2
166 {
167   \exp_args:Nno \use:n
168   {
169     \int_set:Nn \tex_endlinechar:D { -1 }
170     \tex_readline:D #1 to #2
171     \int_set:Nn \tex_endlinechar:D
172     } { \int_use:N \tex_endlinechar:D }
173   }
174 \prg_new_protected_conditional:Npnn \ior_str_get:NN #1#2 { T , F , TF }
175 {
176   \ior_if_eof:NNTF #1
177   { \prg_return_false: }
178   {
179     \_ior_str_get:NN #1 #2
180     \prg_return_true:
181   }
182 }
```

(`\ior_str_get:NN`, `\_ior_str_get:NN`, 和 `\ior_str_get:NNTF` 定义结束。这些函数被记录在第7页。)

`\c_ior_term_noprompt_ior` For reading without a prompt.

```
183 \int_const:Nn \c_ior_term_noprompt_ior { -1 }
```

(`\c_ior_term_noprompt_ior` 定义结束。)

`\ior_get_term:nN` Getting from the terminal is better with pretty-printing.

```
\ior_str_get_term:nN 184 \cs_new_protected:Npn \ior_get_term:nN #1#2
\_ior_get_term:NnN 185 { \_ior_get_term:NnN \_ior_get:NN {#1} #2 }
186 \cs_new_protected:Npn \ior_str_get_term:nN #1#2
187 { \_ior_get_term:NnN \_ior_str_get:NN {#1} #2 }
188 \cs_new_protected:Npn \_ior_get_term:NnN #1#2#3
189 {
190   \group_begin:
191   \tex_escapechar:D = -1 \scan_stop:
192   \tl_if_blank:nTF {#2}
193   { \exp_args:Nnc #1 \c_ior_term_noprompt_ior }
```

```

194         { \exp_args:NNc #1 \c__ior_term_ior }
195         {#2}
196     \exp_args:NNV \group_end:
197     \tl_set:Nn #3 {#2}
198 }

```

(*\ior\_get\_term:nN*, *\ior\_str\_get\_term:nN*, 和 *\\_\_ior\_get\_term:NnN* 定义结束。这些函数被记录在第9页。)

*\ior\_map\_break:* Usual map breaking functions.

```

\ior_map_break:n 199 \cs_new:Npn \ior_map_break:
200   { \prg_map_break:Nn \ior_map_break: { } }
201 \cs_new:Npn \ior_map_break:n
202   { \prg_map_break:Nn \ior_map_break: }

```

(*\ior\_map\_break:* 和 *\ior\_map\_break:n* 定义结束。这些函数被记录在第8页。)

*\ior\_map\_inline:Nn* Mapping over an input stream can be done on either a token or a string basis, hence  
*\ior\_str\_map\_inline:Nn* the set up. Within that, there is a check to avoid reading past the end of a file, hence  
*\\_\_ior\_map\_inline:NNn* the two applications of *\ior\_if\_eof:N* and its lower-level analogue *\if\_eof:w*. This  
*\\_\_ior\_map\_inline:NNNn* mapping cannot be nested with twice the same stream, as the stream has only one  
*\\_\_ior\_map\_inline\_loop:NNN* “current line”.

```

203 \cs_new_protected:Npn \ior_map_inline:Nn
204   { \__ior_map_inline:NNn \__ior_get:NN }
205 \cs_new_protected:Npn \ior_str_map_inline:Nn
206   { \__ior_map_inline:NNn \__ior_str_get:NN }
207 \cs_new_protected:Npn \__ior_map_inline:NNn
208   {
209     \int_gincr:N \g__kernel_prg_map_int
210     \exp_args:Nc \__ior_map_inline:NNNn
211       { __ior_map_ \int_use:N \g__kernel_prg_map_int :n }
212   }
213 \cs_new_protected:Npn \__ior_map_inline:NNNn #1#2#3#4
214   {
215     \cs_gset_protected:Npn #1 ##1 {#4}
216     \ior_if_eof:NF #3 { \__ior_map_inline_loop:NNN #1#2#3 }
217     \prg_break_point:Nn \ior_map_break:
218       { \int_gdecr:N \g__kernel_prg_map_int }
219   }
220 \cs_new_protected:Npn \__ior_map_inline_loop:NNN #1#2#3
221   {
222     #2 #3 \l__ior_internal_tl
223     \if_eof:w #3

```

```

224     \exp_after:wN \ior_map_break:
225     \fi:
226     \exp_args:No #1 \l__ior_internal_tl
227     \__ior_map_inline_loop:NNN #1#2#3
228 }

```

(*\ior\_map\_inline:Nn* 以及其它的定义结束。这些函数被记录在第7页。)

*\ior\_map\_variable:NNn* Since the T<sub>E</sub>X primitive (*\read* or *\readline*) assigns the tokens read in the same way as a token list assignment, we simply call the appropriate primitive. The end-of-loop is checked using the primitive conditional for speed.

```

\ior_str_map_variable:NNn
\__ior_map_variable:NNNn
  \__ior_map_variable_loop:NNNn
229 \cs_new_protected:Npn \ior_map_variable:NNn
230 { \__ior_map_variable:NNNn \ior_get:NN }
231 \cs_new_protected:Npn \ior_str_map_variable:NNn
232 { \__ior_map_variable:NNNn \ior_str_get:NN }
233 \cs_new_protected:Npn \__ior_map_variable:NNNn #1#2#3#4
234 {
235   \ior_if_eof:NF #2 { \__ior_map_variable_loop:NNNn #1#2#3 {#4} }
236   \prg_break_point:Nn \ior_map_break: { }
237 }
238 \cs_new_protected:Npn \__ior_map_variable_loop:NNNn #1#2#3#4
239 {
240   #1 #2 #3
241   \if_eof:w #2
242     \exp_after:wN \ior_map_break:
243     \fi:
244     #4
245     \__ior_map_variable_loop:NNNn #1#2#3 {#4}
246 }

```

(*\ior\_map\_variable:NNn* 以及其它的定义结束。这些函数被记录在第8页。)

## 3.2 Output operations

```

247 <@@=iow>

```

There is a lot of similarity here to the input operations, at least for many of the basics. Thus quite a bit is copied from the earlier material with minor alterations.

### 3.2.1 Variables and constants

*\l\_\_iow\_internal\_tl* Used as a short-term scratch variable.

```

248 \tl_new:N \l__iow_internal_tl

```

(*\l\_\_iow\_internal\_tl* 定义结束。)

`\c_log_iow` Here we allocate two output streams for writing to the transcript file only (`\c_log_iow`) and to both the terminal and transcript file (`\c_term_iow`). Recent LuaTeX provide 128 write streams; we also use `\c_term_iow` as the first non-allowed write stream so its value depends on the engine.

```

249 \int_const:Nn \c_log_iow { -1 }
250 \int_const:Nn \c_term_iow
251 {
252   \bool_lazy_and:nnTF
253     { \sys_if_engine luatex_p: }
254     { \int_compare_p:nNn \tex_luatexversion:D > { 80 } }
255     { 128 }
256     { 16 }
257 }

```

(`\c_log_iow` 和 `\c_term_iow` 定义结束。这些变量被记录在第14页。)

`\g__iow_streams_seq` A list of the currently-available output streams to be used as a stack.

```

258 \seq_new:N \g__iow_streams_seq

```

(`\g__iow_streams_seq` 定义结束。)

`\l__iow_stream_tl` Used to recover the raw stream number from the stack.

```

259 \tl_new:N \l__iow_stream_tl

```

(`\l__iow_stream_tl` 定义结束。)

`\g__iow_streams_prop` As for reads with the appropriate adjustment of the register numbers to check on.

```

260 \prop_new:N \g__iow_streams_prop
261 \int_step_inline:nnn
262 { 0 }
263 {
264   \cs_if_exist:NTF \contextversion
265     { \tex_count:D 39 ~ }
266     {
267       \tex_count:D 17 ~
268       \cs_if_exist:NT \loccount { - 1 }
269     }
270 }
271 {
272   \prop_gput:Nnn \g__iow_streams_prop {#1} { Reserved-by~format }
273 }

```

(`\g__iow_streams_prop` 定义结束。)

### 3.2.2 Internal auxiliaries

`\s__iow_mark` Internal scan marks.

`\s__iow_stop` 274 `\scan_new:N \s__iow_mark`  
275 `\scan_new:N \s__iow_stop`

(`\s__iow_mark` 和 `\s__iow_stop` 定义结束。)

`\_iow_use_i_delimit_by_s_stop:nw` Functions to gobble up to a scan mark.

276 `\cs_new:Npn \_iow_use_i_delimit_by_s_stop:nw #1 #2 \s__iow_stop {#1}`

(`\_iow_use_i_delimit_by_s_stop:nw` 定义结束。)

`\q__iow_nil` Internal quarks.

277 `\quark_new:N \q__iow_nil`

(`\q__iow_nil` 定义结束。)

### 3.3 Stream management

`\iow_new:N` Reserving a new stream is done by defining the name as equal to writing to the terminal: odd but at least consistent.

`\iow_new:c`

278 `\cs_new_protected:Npn \iow_new:N #1 { \cs_new_eq:NN #1 \c_term_iow }`  
279 `\cs_generate_variant:Nn \iow_new:N { c }`

(`\iow_new:N` 定义结束。这个函数被记录在第9页。)

`\g_tmpa_iow` The usual scratch space.

`\g_tmpb_iow` 280 `\iow_new:N \g_tmpa_iow`  
281 `\iow_new:N \g_tmpb_iow`

(`\g_tmpa_iow` 和 `\g_tmpb_iow` 定义结束。这些变量被记录在第14页。)

`\__iow_new:N` As for read streams, copy `\newwrite`, making sure that it is not `\outer`. For ConTeXt, we have to deal with the fact that `\newwrite` works like our own: it actually checks before altering definition.

282 `\exp_args:NNf \cs_new_protected:Npn \__iow_new:N`  
283 `{ \exp_args:NNc \exp_after:wN \exp_stop_f: { newwrite } }`  
284 `\cs_if_exist:NT \contextversion`  
285 `{`  
286 `\cs_new_eq:NN \__iow_new_aux:N \__iow_new:N`  
287 `\cs_gset_protected:Npn \__iow_new:N #1`  
288 `{`  
289 `\cs_undefine:N #1`



```

290         \__iow_new_aux:N #1
291     }
292 }

```

(\\_\_iow\_new:N 定义结束。)

\l\_\_iow\_file\_name\_tl Data storage.

```

293 \tl_new:N \l__iow_file_name_tl

```

(\l\_\_iow\_file\_name\_tl 定义结束。)

\iow\_open:Nn The same idea as for reading, but without the path and without the need to allow  
 \iow\_open:NV for a conditional version.

```

\iow_open:cn 294 \cs_new_protected:Npn \iow_open:Nn #1#2
\iow_open:cV 295 {
\__iow_open_stream:Nn 296     \__kernel_tl_set:Ne \l__iow_file_name_tl
\__iow_open_stream:NV 297     { \__kernel_file_name_sanitiz:n {#2} }
298     \iow_close:N #1
299     \seq_gpop:NNTF \g__iow_streams_seq \l__iow_stream_tl
300     { \__iow_open_stream:NV #1 \l__iow_file_name_tl }
301     {
302         \__iow_new:N #1
303         \__kernel_tl_set:Ne \l__iow_stream_tl { \int_eval:n {#1} }
304         \__iow_open_stream:NV #1 \l__iow_file_name_tl
305     }
306 }
307 \cs_generate_variant:Nn \iow_open:Nn { NV , c , cV }
308 \cs_new_protected:Npn \__iow_open_stream:Nn #1#2
309 {
310     \tex_global:D \tex_chardef:D #1 = \l__iow_stream_tl \scan_stop:
311     \prop_gput:NVn \g__iow_streams_prop #1 {#2}
312     \tex_immediate:D \tex_openout:D
313         #1 \__kernel_file_name_quote:n {#2} \scan_stop:
314 }
315 \cs_generate_variant:Nn \__iow_open_stream:Nn { NV }

```

(\iow\_open:Nn 和 \\_\_iow\_open\_stream:Nn 定义结束。这个函数被记录在第4页。)

\iow\_shell\_open:Nn Very similar to the ior version

```

\__iow_shell_open:nN 316 \cs_new_protected:Npn \iow_shell_open:Nn #1#2
\__iow_shell_open:oN 317 {
318     \sys_if_shell:TF
319     { \__iow_shell_open:oN { \tl_to_str:n {#2} } } #1 }

```

```

320     { \msg_error:nn { kernel } { pipe-failed } }
321   }
322   \cs_new_protected:Npn \__iow_shell_open:nN #1#2
323   {
324     \tl_if_in:nnTF {#1} { " }
325     {
326       \msg_error:nne
327         { kernel } { quote-in-shell } {#1}
328     }
329     { \__kernel_iow_open:Nn #2 { |#1 } }
330   }
331   \cs_generate_variant:Nn \__iow_shell_open:nN { o }

```

(*\iow\_shell\_open:Nn* 和 *\\_\_iow\_shell\_open:nN* 定义结束。这个函数被记录在第5页。)

**\iow\_close:N** Closing a stream is not quite the reverse of opening one. First, the close operation  
**\iow\_close:c** is easier than the open one, and second as the stream is actually a number we can  
use it directly to show that the slot has been freed up.

```

332   \cs_new_protected:Npn \iow_close:N #1
333   {
334     \int_compare:nT { \c_log_iow < #1 < \c_term_iow }
335     {
336       \tex_immediate:D \tex_closeout:D #1
337       \prop_gremove:NV \g__iow_streams_prop #1
338       \seq_if_in:NVF \g__iow_streams_seq #1
339         { \seq_gpush:NV \g__iow_streams_seq #1 }
340       \cs_gset_eq:NN #1 \c_term_iow
341     }
342   }
343   \cs_generate_variant:Nn \iow_close:N { c }

```

(*\iow\_close:N* 定义结束。这个函数被记录在第5页。)

**\iow\_show:N** Seek the stream in the *\g\_\_iow\_streams\_prop* list, then show the stream as open  
**\iow\_log:N** or closed accordingly.

```

\__iow_show:NN 344   \cs_new_protected:Npn \iow_show:N { \__iow_show:NN \tl_show:n }
345   \cs_generate_variant:Nn \iow_show:N { c }
346   \cs_new_protected:Npn \iow_log:N { \__iow_show:NN \tl_log:n }
347   \cs_generate_variant:Nn \iow_log:N { c }
348   \cs_new_protected:Npn \__iow_show:NN #1#2
349   {
350     \__kernel_chk_defined:NT #2
351     {

```

```

352         \prop_get:NVNTF \g__iow_streams_prop #2 \l__iow_internal_tl
353         {
354             \exp_args:Ne #1
355             { \token_to_str:N #2 ~ open: ~ \l__iow_internal_tl }
356         }
357         { \exp_args:Ne #1 { \token_to_str:N #2 ~ closed } }
358     }
359 }

```

(*\iow\_show:N*, *\iow\_log:N*, 和 *\\_\_iow\_show:NN* 定义结束。这些函数被记录在第5页。)

```

\iow_show_list: Done as for input, but with a copy of the auxiliary so the name is correct.
\iow_log_list: 360 \cs_new_protected:Npn \iow_show_list: { \__iow_list:N \msg_show:nneeee }
\__iow_list:N 361 \cs_new_protected:Npn \iow_log_list: { \__iow_list:N \msg_log:nneeee }
362 \cs_new_protected:Npn \__iow_list:N #1
363 {
364     #1 { kernel } { show-streams }
365     { iow }
366     {
367         \prop_map_function:NN \g__iow_streams_prop
368         \msg_show_item_unbraced:nn
369     }
370     { } { }
371 }

```

(*\iow\_show\_list:*, *\iow\_log\_list:*, 和 *\\_\_iow\_list:N* 定义结束。这些函数被记录在第5页。)

### 3.3.1 Deferred writing

```

\iow_shipout_e:Nn First the easy part, this is the primitive, which expects its argument to be braced.
\iow_shipout_e:Ne 372 \cs_new_protected:Npn \iow_shipout_e:Nn #1#2
\iow_shipout_e:cn 373 { \tex_write:D #1 {#2} }
\iow_shipout_e:ce 374 \cs_generate_variant:Nn \iow_shipout_e:Nn { Ne , c, ce }

```

(*\iow\_shipout\_e:Nn* 定义结束。这个函数被记录在第10页。)

```

\iow_shipout:Nn With  $\epsilon$ -TeX available deferred writing without expansion is easy.
\iow_shipout:Ne 375 \cs_new_protected:Npn \iow_shipout:Nn #1#2
\iow_shipout:Nx 376 { \tex_write:D #1 { \exp_not:n {#2} } }
\iow_shipout:cn 377 \cs_generate_variant:Nn \iow_shipout:Nn { Ne , c, ce }
\iow_shipout:ce 378 \cs_generate_variant:Nn \iow_shipout:Nn { Nx , cx }
\iow_shipout:cx (\iow_shipout:Nn 定义结束。这个函数被记录在第10页。)
```

### 3.3.2 Immediate writing

`\__kernel_iow_with:Nnn` If the integer #1 is equal to #2, just leave #3 in the input stream. Otherwise, pass  
`\__iow_with:nNnn` the old value to an auxiliary, which sets the integer to the new value, runs the code,  
`\__iow_with:oNnn` and restores the integer.

```

379 \cs_new_protected:Npn \__kernel_iow_with:Nnn #1#2
380 {
381   \int_compare:nNnTF {#1} = {#2}
382     { \use:n }
383     { \__iow_with:oNnn { \int_use:N #1 } #1 {#2} }
384 }
385 \cs_new_protected:Npn \__iow_with:nNnn #1#2#3#4
386 {
387   \int_set:Nn #2 {#3}
388   #4
389   \int_set:Nn #2 {#1}
390 }
391 \cs_generate_variant:Nn \__iow_with:nNnn { o }

```

(`\__kernel_iow_with:Nnn` 和 `\__iow_with:nNnn` 定义结束。)

`\iow_now:Nn` This routine writes the second argument onto the output stream without expansion.  
`\iow_now:NV` If this stream isn't open, the output goes to the terminal instead. If the first argument  
`\iow_now:Ne` is no output stream at all, we get an internal error. We don't use the expansion done  
`\iow_now:Nx` by `\write` to get the `Nx` variant, because it differs in subtle ways from `x`-expansion,  
`\iow_now:cn` namely, macro parameter characters would not need to be doubled. We set the  
`\iow_now:cV` `\newlinechar` to 10 using `\__kernel_iow_with:Nnn` to support formats such as  
`\iow_now:ce` plain `TEX`: otherwise, `\iow_newline:` would not work. We do not do this for `\iow_-`  
`\iow_now:cx` `shipout:Nn` or `\iow_shipout_x:Nn`, as `TEX` looks at the value of the `\newlinechar`  
at shipout time in those cases.

```

392 \cs_new_protected:Npn \iow_now:Nn #1#2
393 {
394   \__kernel_iow_with:Nnn \tex_newlinechar:D { ^^J }
395   { \tex_immediate:D \tex_write:D #1 { \exp_not:n {#2} } }
396 }
397 \cs_generate_variant:Nn \iow_now:Nn { NV , Ne , c , cV , ce }
398 \cs_generate_variant:Nn \iow_now:Nn { Nx , cx }

```

(`\iow_now:Nn` 定义结束。这个函数被记录在第10页。)

`\iow_log:n` Writing to the log and the terminal directly are relatively easy; as we need the two  
`\iow_log:e` e-type variants for bootstrapping, they are redefinitions here.

`\iow_log:x`  
`\iow_term:n`  
`\iow_term:e`  
`\iow_term:x`

```

399 \cs_new_protected:Npn \iow_log:n { \iow_now:Nn \c_log_iow }
400 \cs_set_protected:Npn \iow_log:e { \iow_now:Ne \c_log_iow }
401 \cs_generate_variant:Nn \iow_log:n { x }
402 \cs_new_protected:Npn \iow_term:n { \iow_now:Nn \c_term_iow }
403 \cs_set_protected:Npn \iow_term:e { \iow_now:Ne \c_term_iow }
404 \cs_generate_variant:Nn \iow_term:n { x }

```

(`\iow_log:n` 和 `\iow_term:n` 定义结束。这些函数被记录在第10页。)

### 3.3.3 Special characters for writing

`\iow_newline:` Global variable holding the character that forces a new line when something is written to an output stream.

```

405 \cs_new:Npn \iow_newline: { ^^J }

```

(`\iow_newline:` 定义结束。这个函数被记录在第11页。)

`\iow_char:N` Function to write any escaped char to an output stream.

```

406 \cs_new_eq:NN \iow_char:N \cs_to_str:N

```

(`\iow_char:N` 定义结束。这个函数被记录在第11页。)

### 3.3.4 Hard-wrapping lines to a character count

The code here implements a generic hard-wrapping function. This is used by the messaging system, but is designed such that it is available for other uses.

`\l_iow_line_count_int` This is the “raw” number of characters in a line which can be written to the terminal. The standard value is the line length typically used by T<sub>E</sub>X Live and MiK<sub>T</sub>E<sub>X</sub>.

```

407 \int_new:N \l_iow_line_count_int
408 \int_set:Nn \l_iow_line_count_int { 78 }

```

(`\l_iow_line_count_int` 定义结束。这个变量被记录在第14页。)

`\l__iow_newline_tl` The token list inserted to produce a new line, with the *⟨run-on text⟩*.

```

409 \tl_new:N \l__iow_newline_tl

```

(`\l__iow_newline_tl` 定义结束。)

`\l__iow_line_target_int` This stores the target line count: the full number of characters in a line, minus any part for a leader at the start of each line.

```

410 \int_new:N \l__iow_line_target_int

```

(`\l__iow_line_target_int` 定义结束。)

`\__iow_set_indent:n` The `one_indent` variables hold one indentation marker and its length. The `\__iow_unindent:w` auxiliary removes one indentation. The function `\__iow_set_indent:n` (that could possibly be public) sets the indentation in a consistent way. We set it to four spaces by default.

```

411 \tl_new:N \l__iow_one_indent_tl
412 \int_new:N \l__iow_one_indent_int
413 \cs_new:Npn \__iow_unindent:w { }
414 \cs_new_protected:Npn \__iow_set_indent:n #1
415 {
416   \__kernel_tl_set:Ne \l__iow_one_indent_tl
417   { \exp_args:No \__kernel_str_to_other_fast:n { \tl_to_str:n {#1} } } }
418   \int_set:Nn \l__iow_one_indent_int
419   { \str_count:N \l__iow_one_indent_tl }
420   \exp_last_unbraced:NNo
421   \cs_set:Npn \__iow_unindent:w \l__iow_one_indent_tl { }
422 }
423 \exp_args:Ne \__iow_set_indent:n { \prg_replicate:nn { 4 } { ~ } }

```

(`\__iow_set_indent:n` 以及其它的定义结束。)

`\l__iow_indent_tl` The current indentation (some copies of `\l__iow_one_indent_tl`) and its number of characters.

```

424 \tl_new:N \l__iow_indent_tl
425 \int_new:N \l__iow_indent_int

```

(`\l__iow_indent_tl` 和 `\l__iow_indent_int` 定义结束。)

`\l__iow_line_tl` These hold the current line of text and a partial line to be added to it, respectively.

```

426 \tl_new:N \l__iow_line_tl
427 \tl_new:N \l__iow_line_part_tl

```

(`\l__iow_line_tl` 和 `\l__iow_line_part_tl` 定义结束。)

`\l__iow_line_break_bool` Indicates whether the line was broken precisely at a chunk boundary.

```

428 \bool_new:N \l__iow_line_break_bool

```

(`\l__iow_line_break_bool` 定义结束。)

`\l__iow_wrap_tl` Used for the expansion step before detokenizing, and for the output from wrapping text: fully expanded and with lines which are not overly long.

```

429 \tl_new:N \l__iow_wrap_tl

```

(`\l__iow_wrap_tl` 定义结束。)

`\c__iow_wrap_marker_tl` Every special action of the wrapping code is starts with the same recognizable string, `\c__iow_wrap_marker_tl`. Upon seeing that “word”, the wrapping code reads one space-delimited argument to know what operation to perform. The setting of `\escapechar` here is not very important, but makes `\c__iow_wrap_marker_tl` look marginally nicer.

```

430 \group_begin:
431   \int_set:Nn \tex_escapechar:D { -1 }
432   \tl_const:Nc \c__iow_wrap_marker_tl
433     { \tl_to_str:n { \^^I \^^O \^^W \^^_ \^^W \^^R \^^A \^^P } }
434 \group_end:
435 \tl_map_inline:nn
436   { { end } { newline } { allow_break } { indent } { unindent } }
437   {
438     \tl_const:ce { c__iow_wrap_ #1 _marker_tl }
439     {
440       \c__iow_wrap_marker_tl
441       #1
442       \c_catcode_other_space_tl
443     }
444   }

```

(`\c__iow_wrap_marker_tl` 以及其它的定义结束。)

`\iow_wrap_allow_break:` We set `\iow_wrap_allow_break:n` to produce an error when outside messages.  
`\__iow_wrap_allow_break:` Within wrapped message, it is set to `\__iow_wrap_allow_break:` when valid and  
`\__iow_wrap_allow_break_error:` otherwise to `\__iow_wrap_allow_break_error:.` The second produces an error expandably.

```

445 \cs_new_protected:Npn \iow_wrap_allow_break:
446   {
447     \msg_error:nnnn { kernel } { iow-indent }
448     { \iow_wrap:nnnN } { \iow_wrap_allow_break: }
449   }
450 \cs_new:Npe \__iow_wrap_allow_break: { \c__iow_wrap_allow_break_marker_tl }
451 \cs_new:Npn \__iow_wrap_allow_break_error:
452   {
453     \msg_expandable_error:nnnn { kernel } { iow-indent }
454     { \iow_wrap:nnnN } { \iow_wrap_allow_break: }
455   }

```

(`\iow_wrap_allow_break:`, `\__iow_wrap_allow_break:`, 和 `\__iow_wrap_allow_break_error:` 定义结束。这个函数被记录在第 14 页。)

`\iow_indent:n` We set `\iow_indent:n` to produce an error when outside messages. Within wrapped message, it is set to `\__iow_indent:n` when valid and otherwise to `\__iow_indent_error:n`. The first places the instruction for increasing the indentation before its argument, and the instruction for unindenting afterwards. The second produces an error expandably. Note that there are no forced line-break, so the indentation only changes when the next line is started.

```

456 \cs_new_protected:Npn \iow_indent:n #1
457 {
458   \msg_error:nnnnn { kernel } { iow-indent }
459   { \iow_wrap:nnnN } { \iow_indent:n } {#1}
460   #1
461 }
462 \cs_new:Npe \__iow_indent:n #1
463 {
464   \c__iow_wrap_indent_marker_tl
465   #1
466   \c__iow_wrap_unindent_marker_tl
467 }
468 \cs_new:Npn \__iow_indent_error:n #1
469 {
470   \msg_expandable_error:nnnnn { kernel } { iow-indent }
471   { \iow_wrap:nnnN } { \iow_indent:n } {#1}
472   #1
473 }
```

(`\iow_indent:n`, `\__iow_indent:n`, 和 `\__iow_indent_error:n` 定义结束。这个函数被记录在第14页。)

`\iow_wrap:nnnN` The main wrapping function works as follows. First give `\`, `\_` and other formatting commands the correct definition for messages and perform the given setup #3. `\iow_wrap:nenN` The definition of `\_` uses an “other” space rather than a normal space, because the latter might be absorbed by  $\text{\TeX}$  to end a number or other f-type expansions. Use `\conditionally@traceoff` if defined; it is introduced by the `trace` package and suppresses uninteresting tracing of the wrapping code.

```

474 \cs_new_protected:Npn \iow_wrap:nnnN #1#2#3#4
475 {
476   \group_begin:
477   \cs_if_exist_use:N \conditionally@traceoff
478   \int_set:Nn \tex_escapechar:D { -1 }
479   \cs_set:Npe \{ { \token_to_str:N \{ }
480   \cs_set:Npe \# { \token_to_str:N \# }
481   \cs_set:Npe \} { \token_to_str:N \} }
```



```

482     \cs_set:Npe \% { \token_to_str:N \% }
483     \cs_set:Npe \~ { \token_to_str:N \~ }
484     \int_set:Nn \tex_escapechar:D { 92 }
485     \cs_set_eq:NN \\ \iow_newline:
486     \cs_set_eq:NN \ \c_catcode_other_space_tl
487     \cs_set_eq:NN \iow_wrap_allow_break: \__iow_wrap_allow_break:
488     \cs_set_eq:NN \iow_indent:n \__iow_indent:n
489     #3

```

Then fully-expand the input: in package mode, the expansion uses L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>’s `\protect` mechanism in the same way as `\typeout`. In generic mode this setting is useless but harmless. As soon as the expansion is done, reset `\iow_indent:n` to its error definition: it only works in the first argument of `\iow_wrap:nnnN`.

```

490     \cs_set_eq:NN \protect \token_to_str:N
491     \__kernel_tl_set:Ne \l__iow_wrap_tl {#1}
492     \cs_set_eq:NN \iow_wrap_allow_break: \__iow_wrap_allow_break_error:
493     \cs_set_eq:NN \iow_indent:n \__iow_indent_error:n

```

Afterwards, set the newline marker (two assignments to fully expand, then convert to a string) and initialize the target count for lines (the first line has target count `\l_iow_line_count_int` instead).

```

494     \__kernel_tl_set:Ne \l__iow_newline_tl { \iow_newline: #2 }
495     \__kernel_tl_set:Ne \l__iow_newline_tl { \tl_to_str:N \l__iow_newline_tl }
496     \int_set:Nn \l__iow_line_target_int
497     { \l_iow_line_count_int - \str_count:N \l__iow_newline_tl + 1 }

```

Sanity check.

```

498     \int_compare:nNnT { \l__iow_line_target_int } < 0
499     {
500         \tl_set:Nn \l__iow_newline_tl { \iow_newline: }
501         \int_set:Nn \l__iow_line_target_int
502         { \l_iow_line_count_int + 1 }
503     }

```

There is then a loop over the input, which stores the wrapped result in `\l__iow_wrap_tl`. After the loop, the resulting text is passed on to the function which has been given as a post-processor. The `\tl_to_str:N` step converts the “other” spaces back to normal spaces. The f-expansion removes a leading space from `\l__iow_wrap_tl`.

```

504     \__iow_wrap_do:
505     \exp_args:NNf \group_end:
506     #4 { \tl_to_str:N \l__iow_wrap_tl }
507 }

```

```
508 \cs_generate_variant:Nn \iow_wrap:nnnN { ne }
```

(*\iow\_wrap:nnnN* 定义结束。这个函数被记录在第13页。)

**\\_\_iow\_wrap\_do:** Escape spaces and change newlines to `\c__iow_wrap_newline_marker_tl`. Set up  
**\\_\_iow\_wrap\_fix\_newline:w** a few variables, in particular the initial value of `\l__iow_wrap_tl`: the space stops  
**\\_\_iow\_wrap\_start:w** the f-expansion of the main wrapping function and `\use_none:n` removes a newline  
marker inserted by later code. The main loop consists of repeatedly calling the `chunk`  
auxiliary to wrap chunks delimited by (newline or indentation) markers.

```
509 \cs_new_protected:Npn \__iow_wrap_do:
510 {
511   \__kernel_tl_set:Ne \l__iow_wrap_tl
512   {
513     \exp_args:No \__kernel_str_to_other_fast:n \l__iow_wrap_tl
514     \c__iow_wrap_end_marker_tl
515   }
516   \__kernel_tl_set:Ne \l__iow_wrap_tl
517   {
518     \exp_after:wN \__iow_wrap_fix_newline:w \l__iow_wrap_tl
519     ^^J \q__iow_nil ^^J \s__iow_stop
520   }
521   \exp_after:wN \__iow_wrap_start:w \l__iow_wrap_tl
522 }
523 \cs_new:Npn \__iow_wrap_fix_newline:w #1 ^^J #2 ^^J
524 {
525   #1
526   \if_meaning:w \q__iow_nil #2
527   \__iow_use_i_delimit_by_s_stop:nw
528   \fi:
529   \c__iow_wrap_newline_marker_tl
530   \__iow_wrap_fix_newline:w #2 ^^J
531 }
532 \cs_new_protected:Npn \__iow_wrap_start:w
533 {
534   \bool_set_false:N \l__iow_line_break_bool
535   \tl_clear:N \l__iow_line_tl
536   \tl_clear:N \l__iow_line_part_tl
537   \tl_set:Nn \l__iow_wrap_tl { ~ \use_none:n }
538   \int_zero:N \l__iow_indent_int
539   \tl_clear:N \l__iow_indent_tl
540   \__iow_wrap_chunk:nw { \l__iow_line_count_int }
541 }
```

(`\_iow_wrap_do:`, `\_iow_wrap_fix_newline:w`, 和 `\_iow_wrap_start:w` 定义结束。)

`\_iow_wrap_chunk:nw` The `chunk` and `next` auxiliaries are defined indirectly to obtain the expansions of `\c_catcode_other_space_tl` and `\c__iow_wrap_marker_tl` in their definition. `\_iow_wrap_next:nw` The `next` auxiliary calls a function corresponding to the type of marker (its `##2`), which can be `newline` or `indent` or `unindent` or `end`. The first argument of the `chunk` auxiliary is a target number of characters and the second is some string to wrap. If the chunk is empty simply call `next`. Otherwise, set up a call to `\_iow_wrap_line:nw`, including the indentation if the current line is empty, and including a trailing space (`#1`) before the `\_iow_wrap_end_chunk:w` auxiliary.

```

542 \cs_set_protected:Npn \_iow_tmp:w #1#2
543 {
544   \cs_new_protected:Npn \_iow_wrap_chunk:nw ##1##2 #2
545   {
546     \tl_if_empty:nTF {##2}
547     {
548       \tl_clear:N \l__iow_line_part_tl
549       \_iow_wrap_next:nw {##1}
550     }
551     {
552       \tl_if_empty:NTF \l__iow_line_tl
553       {
554         \_iow_wrap_line:nw
555         { \l__iow_indent_tl }
556         ##1 - \l__iow_indent_int ;
557       }
558       { \_iow_wrap_line:nw { } ##1 ; }
559       ##2 #1
560       \_iow_wrap_end_chunk:w 7 6 5 4 3 2 1 0 \s__iow_stop
561     }
562   }
563   \cs_new_protected:Npn \_iow_wrap_next:nw ##1##2 #1
564   { \use:c { __iow_wrap_##2:n } {##1} }
565 }
566 \exp_args:NVV \_iow_tmp:w \c_catcode_other_space_tl \c__iow_wrap_marker_tl

```

(`\_iow_wrap_chunk:nw` 和 `\_iow_wrap_next:nw` 定义结束。)

`\_iow_wrap_line:nw` This is followed by `{⟨string⟩}⟨int expr⟩ ;`. It stores the `⟨string⟩` and up to `⟨int expr⟩` characters from the current chunk into `\l__iow_line_part_tl`. Characters are grabbed 8 at a time and left in `\l__iow_line_part_tl` by the `line_loop` auxiliary.

`\_iow_wrap_line_loop:w`

`\_iow_wrap_line_aux:Nw`

`\_iow_wrap_line_seven:nnnnnnn`

`\_iow_wrap_line_end:NnnnnnnnN`

`\_iow_wrap_line_end:nw`

`\_iow_wrap_end_chunk:w`

When  $k < 8$  remain to be found, the `line_aux` auxiliary calls the `line_end` auxiliary followed by (the single digit)  $k$ , then  $7 - k$  empty brace groups, then the chunk's remaining characters. The `line_end` auxiliary leaves  $k$  characters from the chunk in the line part, then ends the assignment. Ignore the `\use_none:nnnnn` line for now. If the next character is a space the line can be broken there: store what we found into the result and get the next line. Otherwise some work is needed to find a break-point. So far we have ignored what happens if the chunk is shorter than the requested number of characters: this is dealt with by the `end_chunk` auxiliary, which gets treated like a character by the rest of the code. It ends up being called either as one of the arguments #2–#9 of the `line_loop` auxiliary or as one of the arguments #2–#8 of the `line_end` auxiliary. In both cases stop the assignment and work out how many characters are still needed. Notice that when we have exactly seven arguments to clean up, a `\exp_stop_f:` has to be inserted to stop the `\exp:w`. The weird `\use_none:nnnnn` ensures that the required data is in the right place.

```

567 \cs_new_protected:Npn \__iow_wrap_line:nw #1
568 {
569   \tex_edef:D \l__iow_line_part_tl { \if_false: } \fi:
570   #1
571   \exp_after:wN \__iow_wrap_line_loop:w
572   \int_value:w \int_eval:w
573 }
574 \cs_new:Npn \__iow_wrap_line_loop:w #1 ; #2#3#4#5#6#7#8#9
575 {
576   \if_int_compare:w #1 < 8 \exp_stop_f:
577     \__iow_wrap_line_aux:Nw #1
578   \fi:
579   #2 #3 #4 #5 #6 #7 #8 #9
580   \exp_after:wN \__iow_wrap_line_loop:w
581   \int_value:w \int_eval:w #1 - 8 ;
582 }
583 \cs_new:Npn \__iow_wrap_line_aux:Nw #1#2#3 \exp_after:wN #4 ;
584 {
585   #2
586   \exp_after:wN \__iow_wrap_line_end:NnnnnnnnN
587   \exp_after:wN #1
588   \exp:w \exp_end_continue_f:w
589   \exp_after:wN \exp_after:wN
590   \if_case:w #1 \exp_stop_f:
591     \prg_do_nothing:
592   \or: \use_none:n

```

```

593     \or: \use_none:nn
594     \or: \use_none:nnn
595     \or: \use_none:nnnn
596     \or: \use_none:nnnnn
597     \or: \use_none:nnnnnn
598     \or: \__iow_wrap_line_seven:nnnnnnn
599     \fi:
600     { } { } { } { } { } { } { } { } { } #3
601   }
602   \cs_new:Npn \__iow_wrap_line_seven:nnnnnnn #1#2#3#4#5#6#7 { \exp_stop_f: }
603   \cs_new:Npn \__iow_wrap_line_end:NnnnnnnnN #1#2#3#4#5#6#7#8#9
604   {
605     #2 #3 #4 #5 #6 #7 #8
606     \use_none:nnnnn \int_eval:w 8 - ; #9
607     \token_if_eq_charcode:NNTF \c_space_token #9
608     { \__iow_wrap_line_end:nw { } }
609     { \if_false: { \fi: } \__iow_wrap_break:w #9 }
610   }
611   \cs_new:Npn \__iow_wrap_line_end:nw #1
612   {
613     \if_false: { \fi: }
614     \__iow_wrap_store_do:n {#1}
615     \__iow_wrap_next_line:w
616   }
617   \cs_new:Npn \__iow_wrap_end_chunk:w
618   #1 \int_eval:w #2 - #3 ; #4#5 \s__iow_stop
619   {
620     \if_false: { \fi: }
621     \exp_args:Nf \__iow_wrap_next:nw { \int_eval:n { #2 - #4 } }
622   }

```

(*\\_\_iow\_wrap\_line:nw* 以及其它的定义结束。)

*\\_\_iow\_wrap\_break:w* Functions here are defined indirectly: *\\_\_iow\_tmp:w* is eventually called with an “other” space as its argument. The goal is to remove from *\l\_\_iow\_line\_part\_tl* the part after the last space. In most cases this is done by repeatedly calling the *\\_\_iow\_wrap\_break\_first:w* *break\_loop* auxiliary, which leaves “words” (delimited by spaces) until it hits the trailing space: then its argument *##3* is ? *\\_\_iow\_wrap\_break\_end:w* instead of a single token, and that *break\_end* auxiliary leaves in the assignment the line until the last space, then calls *\\_\_iow\_wrap\_line\_end:nw* to finish up the line and move on to the next. If there is no space in *\l\_\_iow\_line\_part\_tl* then the *break\_first* auxiliary calls the *break\_none* auxiliary. In that case, if the current line is empty,

the complete word (including ##4, characters beyond what we had grabbed) is added to the line, making it over-long. Otherwise, the word is used for the following line (and the last space of the line so far is removed because it was inserted due to the presence of a marker).

```

623 \cs_set_protected:Npn \__iow_tmp:w #1
624 {
625   \cs_new:Npn \__iow_wrap_break:w
626   {
627     \tex_edef:D \l__iow_line_part_tl
628     { \if_false: } \fi:
629     \exp_after:wN \__iow_wrap_break_first:w
630     \l__iow_line_part_tl
631     #1
632     { ? \__iow_wrap_break_end:w }
633     \s__iow_mark
634   }
635   \cs_new:Npn \__iow_wrap_break_first:w ##1 #1 ##2
636   {
637     \use_none:nn ##2 \__iow_wrap_break_none:w
638     \__iow_wrap_break_loop:w ##1 #1 ##2
639   }
640   \cs_new:Npn \__iow_wrap_break_none:w ##1##2 #1 ##3 \s__iow_mark ##4 #1
641   {
642     \tl_if_empty:NTF \l__iow_line_tl
643     { ##2 ##4 \__iow_wrap_line_end:nw { } }
644     { \__iow_wrap_line_end:nw { \__iow_wrap_trim:N } ##2 ##4 #1 }
645   }
646   \cs_new:Npn \__iow_wrap_break_loop:w ##1 #1 ##2 #1 ##3
647   {
648     \use_none:n ##3
649     ##1 #1
650     \__iow_wrap_break_loop:w ##2 #1 ##3
651   }
652   \cs_new:Npn \__iow_wrap_break_end:w ##1 #1 ##2 ##3 #1 ##4 \s__iow_mark
653   { ##1 \__iow_wrap_line_end:nw { } ##3 }
654 }
655 \exp_args:NV \__iow_tmp:w \c_catcode_other_space_tl

```

(`\__iow_wrap_break:w` 以及其它的定义结束。)

`\__iow_wrap_next_line:w` The special case where the end of a line coincides with the end of a chunk is detected here, to avoid a spurious empty line. Otherwise, call `\__iow_wrap_line:nw` to find

characters for the next line (remembering to account for the indentation).

```

656 \cs_new_protected:Npn \__iow_wrap_next_line:w #1#2 \s__iow_stop
657 {
658   \tl_clear:N \l__iow_line_tl
659   \token_if_eq_meaning:NNTF #1 \__iow_wrap_end_chunk:w
660   {
661     \tl_clear:N \l__iow_line_part_tl
662     \bool_set_true:N \l__iow_line_break_bool
663     \__iow_wrap_next:nw { \l__iow_line_target_int }
664   }
665   {
666     \__iow_wrap_line:nw
667     { \l__iow_indent_tl }
668     \l__iow_line_target_int - \l__iow_indent_int ;
669     #1 #2 \s__iow_stop
670   }
671 }

```

(`\__iow_wrap_next_line:w` 定义结束。)

`\__iow_wrap_allow_break:n` This is called after a chunk has been wrapped. The `\l__iow_line_part_tl` typically ends with a space (except at the beginning of a line?), which we remove since the `allow_break` marker should not insert a space. Then move on with the next chunk, making sure to adjust the target number of characters for the line in case we did remove a space.

```

672 \cs_new_protected:Npn \__iow_wrap_allow_break:n #1
673 {
674   \__kernel_tl_set:Ne \l__iow_line_tl
675   { \l__iow_line_tl \__iow_wrap_trim:N \l__iow_line_part_tl }
676   \bool_set_false:N \l__iow_line_break_bool
677   \tl_if_empty:NNTF \l__iow_line_part_tl
678   { \__iow_wrap_chunk:nw {#1} }
679   { \exp_args:Nf \__iow_wrap_chunk:nw { \int_eval:n { #1 + 1 } } }
680 }

```

(`\__iow_wrap_allow_break:n` 定义结束。)

`\__iow_wrap_indent:n` These functions are called after a chunk has been wrapped, when encountering `indent/unindent` markers. Add the line part (last line part of the previous chunk) to the line so far and reset a boolean denoting the presence of a line-break. Most importantly, add or remove one indent from the current indent (both the integer and the token list). Finally, continue wrapping.

```

681 \cs_new_protected:Npn \__iow_wrap_indent:n #1
682 {
683   \tl_put_right:Ne \l__iow_line_tl { \l__iow_line_part_tl }
684   \bool_set_false:N \l__iow_line_break_bool
685   \int_add:Nn \l__iow_indent_int { \l__iow_one_indent_int }
686   \tl_put_right:No \l__iow_indent_tl { \l__iow_one_indent_tl }
687   \__iow_wrap_chunk:nw {#1}
688 }
689 \cs_new_protected:Npn \__iow_wrap_unindent:n #1
690 {
691   \tl_put_right:Ne \l__iow_line_tl { \l__iow_line_part_tl }
692   \bool_set_false:N \l__iow_line_break_bool
693   \int_sub:Nn \l__iow_indent_int { \l__iow_one_indent_int }
694   \__kernel_tl_set:Ne \l__iow_indent_tl
695     { \exp_after:wN \__iow_unindent:w \l__iow_indent_tl }
696   \__iow_wrap_chunk:nw {#1}
697 }

```

(`\__iow_wrap_indent:n` 和 `\__iow_wrap_unindent:n` 定义结束。)

`\__iow_wrap_newline:n` These functions are called after a chunk has been line-wrapped, when encountering a **newline**/end marker. Unless we just took a line-break, store the line part and the line so far into the whole `\l__iow_wrap_tl`, trimming a trailing space. In the **newline** case look for a new line (of length `\l__iow_line_target_int`) in a new chunk.

```

698 \cs_new_protected:Npn \__iow_wrap_newline:n #1
699 {
700   \bool_if:NF \l__iow_line_break_bool
701     { \__iow_wrap_store_do:n { \__iow_wrap_trim:N } }
702   \bool_set_false:N \l__iow_line_break_bool
703   \__iow_wrap_chunk:nw { \l__iow_line_target_int }
704 }
705 \cs_new_protected:Npn \__iow_wrap_end:n #1
706 {
707   \bool_if:NF \l__iow_line_break_bool
708     { \__iow_wrap_store_do:n { \__iow_wrap_trim:N } }
709   \bool_set_false:N \l__iow_line_break_bool
710 }

```

(`\__iow_wrap_newline:n` 和 `\__iow_wrap_end:n` 定义结束。)

`\__iow_wrap_store_do:n` First add the last line part to the line, then append it to `\l__iow_wrap_tl` with the appropriate new line (with “run-on” text), possibly with its last space removed (#1



is empty or `\__iow_wrap_trim:N`).

```

711 \cs_new_protected:Npn \__iow_wrap_store_do:n #1
712 {
713   \__kernel_tl_set:Ne \l__iow_line_tl
714   { \l__iow_line_tl \l__iow_line_part_tl }
715   \__kernel_tl_set:Ne \l__iow_wrap_tl
716   {
717     \l__iow_wrap_tl
718     \l__iow_newline_tl
719     #1 \l__iow_line_tl
720   }
721   \tl_clear:N \l__iow_line_tl
722 }

```

(`\__iow_wrap_store_do:n` 定义结束。)

`\__iow_wrap_trim:N` Remove one trailing “other” space from the argument if present.

```

\__iow_wrap_trim:w 723 \cs_set_protected:Npn \__iow_tmp:w #1
\__iow_wrap_trim_aux:w 724 {
725   \cs_new:Npn \__iow_wrap_trim:N ##1
726   { \exp_after:wN \__iow_wrap_trim:w ##1 \s__iow_mark #1 \s__iow_mark \s__iow_stop }
727   \cs_new:Npn \__iow_wrap_trim:w ##1 #1 \s__iow_mark
728   { \__iow_wrap_trim_aux:w ##1 \s__iow_mark }
729   \cs_new:Npn \__iow_wrap_trim_aux:w ##1 \s__iow_mark ##2 \s__iow_stop {##1}
730 }
731 \exp_args:NV \__iow_tmp:w \c_catcode_other_space_tl

```

(`\__iow_wrap_trim:N`, `\__iow_wrap_trim:w`, 和 `\__iow_wrap_trim_aux:w` 定义结束。)

732 `<@@=file>`

### 3.4 File operations

`\l__file_internal_tl` Used as a short-term scratch variable.

```

733 \tl_new:N \l__file_internal_tl

```

(`\l__file_internal_tl` 定义结束。)

`\g_file_curr_dir_str` The name of the current file should be available at all times: the name itself is set dynamically.

```

\g_file_curr_ext_str
\g_file_curr_name_str 734 \str_new:N \g_file_curr_dir_str
735 \str_new:N \g_file_curr_ext_str
736 \str_new:N \g_file_curr_name_str

```

(`\g_file_curr_dir_str`, `\g_file_curr_ext_str`, 和 `\g_file_curr_name_str` 定义结束。这些变量被记录在第15页。)

`\g__file_stack_seq` The input list of files is stored as a sequence stack. In package mode we can recover the information from the details held by  $\text{\LaTeX}2_{\epsilon}$  (we must be in the preamble and loaded using `\usepackage` or `\RequirePackage`). As  $\text{\LaTeX}2_{\epsilon}$  doesn't store directory and name separately, we stick to the same convention here. In pre-loading, `\@currnamestack` is empty so is skipped.

```

737 \seq_new:N \g__file_stack_seq
738 \group_begin:
739   \cs_set_protected:Npn \__file_tmp:w #1#2#3
740   {
741     \tl_if_blank:nTF {#1}
742     {
743       \cs_set:Npn \__file_tmp:w ##1 " ##2 " ##3 \s__file_stop
744       { { } {##2} { } }
745       \seq_gput_right:Ne \g__file_stack_seq
746       {
747         \exp_after:wN \__file_tmp:w \tex_jobname:D
748         " \tex_jobname:D " \s__file_stop
749       }
750     }
751     {
752       \seq_gput_right:Nn \g__file_stack_seq { { } {#1} {#2} }
753       \__file_tmp:w
754     }
755   }
756   \cs_if_exist:NT \@currnamestack
757   {
758     \tl_if_empty:NF \@currnamestack
759     { \exp_after:wN \__file_tmp:w \@currnamestack }
760   }
761 \group_end:

```

(`\g__file_stack_seq` 定义结束。)

`\g__file_record_seq` The total list of files used is recorded separately from the current file stack, as nothing is ever popped from this list. The current file name should be included in the file list! We will eventually copy the contents of `\@filelist`.

```

762 \seq_new:N \g__file_record_seq

```

(`\g__file_record_seq` 定义结束。)

`\l__file_base_name_tl` For storing the basename and full path whilst passing data internally.

`\l__file_full_name_tl` 763 `\tl_new:N \l__file_base_name_tl`

764 `\tl_new:N \l__file_full_name_tl`

(`\l__file_base_name_tl` 和 `\l__file_full_name_tl` 定义结束。)

`\l__file_dir_str` Used in parsing a path into parts: in contrast to the above, these are never used  
`\l__file_ext_str` outside of the current module.

`\l__file_name_str` 765 `\str_new:N \l__file_dir_str`

766 `\str_new:N \l__file_ext_str`

767 `\str_new:N \l__file_name_str`

(`\l__file_dir_str`, `\l__file_ext_str`, 和 `\l__file_name_str` 定义结束。)

`\l_file_search_path_seq` The current search path.

768 `\seq_new:N \l_file_search_path_seq`

(`\l_file_search_path_seq` 定义结束。这个变量被记录在第15页。)

`\l__file_tmp_seq` Scratch space for comma list conversion.

769 `\seq_new:N \l__file_tmp_seq`

(`\l__file_tmp_seq` 定义结束。)

### 3.4.1 Internal auxiliaries

`\s__file_stop` Internal scan marks.

770 `\scan_new:N \s__file_stop`

(`\s__file_stop` 定义结束。)

`\q__file_nil` Internal quarks.

771 `\quark_new:N \q__file_nil`

(`\q__file_nil` 定义结束。)

`\__file_quark_if_nil_p:n` Branching quark conditional.

`\__file_quark_if_nil:nTF` 772 `\__kernel_quark_new_conditional:Nn \__file_quark_if_nil:n { TF }`

(`\__file_quark_if_nil:nTF` 定义结束。)

`\q__file_recursion_tail` Internal recursion quarks.

`\q__file_recursion_stop` 773 `\quark_new:N \q__file_recursion_tail`

774 `\quark_new:N \q__file_recursion_stop`

(`\q__file_recursion_tail` 和 `\q__file_recursion_stop` 定义结束。)

```
\_file_if_recursion_tail_break:Nw Functions to query recursion quarks.
\_file_if_recursion_tail_stop_do:Nn 775 \__kernel_quark_new_test:N \__file_if_recursion_tail_stop:N
776 \__kernel_quark_new_test:N \__file_if_recursion_tail_stop_do:nn
```

(`\__file_if_recursion_tail_break:NN` 和 `\__file_if_recursion_tail_stop_do:Nn` 定义结束。)

```
\_kernel_file_name_sanitize:n Expanding the file name uses a \csname-based approach, and relies on active char-
\_file_name_expand:n characters (for example from UTF-8 characters) being properly set up to expand to a
\_file_name_expand_cleanup:Nw expansion-safe version using \ifcsname. This is less conservative than the token-
\_file_name_expand_cleanup:w by-token approach used before, but it is much faster.
\_file_name_expand_end: 777 \cs_new:Npn \__kernel_file_name_sanitize:n #1
\_file_name_expand_error:Nw 778 {
\_file_name_expand_error_aux:Nw 779 \exp_args:Ne \__file_name_trim_spaces:n
780 {
\_file_name_strip_quotes:n 781 \exp_args:Ne \__file_name_strip_quotes:n
782 { \__file_name_expand:n {#1} }
\_file_name_strip_quotes:nnn 783 }
\_file_name_strip_quotes:nnn 784 }
\_file_name_trim_spaces:n
\_file_name_trim_spaces:nw
```

We'll use `\cs:w` to start expanding the file name, and to avoid creating csnames equal to `\relax` with “common” names, there's a prefix `__file_name=` to the csname. There's also a guard token at the end so we can check if there was an error during the process and (try to) clean up gracefully.

```
785 \cs_new:Npn \__file_name_expand:n #1
786 {
787 \exp_after:wN \__file_name_expand_cleanup:Nw
788 \cs:w __file_name = #1 \cs_end:
789 \__file_name_expand_end:
790 }
```

With the csname built, we grab it, and grab the remaining tokens delimited by `\__file_name_expand_end:.` If there are any remaining tokens, something bad happened, so we'll call the error procedure `\__file_name_expand_error:Nw`. If everything went according to plan, then use `\token_to_str:N` on the csname built, and call `\__file_name_expand_cleanup:w` to remove the prefix we added a while back. `\__file_name_expand_cleanup:w` takes a leading argument so we don't have to bother about the value of `\tex_escapechar:D`.

```
791 \cs_new:Npn \__file_name_expand_cleanup:Nw #1 #2 \__file_name_expand_end:
792 {
793 \tl_if_empty:nF {#2}
```

```

794     { \__file_name_expand_error:Nw #2 \__file_name_expand_end: }
795     \exp_after:wN \__file_name_expand_cleanup:w \token_to_str:N #1
796   }
797   \exp_last_unbraced:NNNNo
798   \cs_new:Npn \__file_name_expand_cleanup:w #1 \tl_to_str:n { __file_name = } { }

```

In non-error cases `\__file_name_expand_end:` should not expand. It will only do so in case there is a `\csname` too much in the file name, so it will throw an error (while expanding), then insert the missing `\cs_end:` and yet another `\__file_name_expand_end:` that will be used as a delimiter by `\__file_name_expand_cleanup:Nw` (or that will expand again if yet another `\endcsname` is missing).

```

799   \cs_new:Npn \__file_name_expand_end:
800   {
801     \msg_expandable_error:nn
802       { kernel } { filename-missing-endcsname }
803     \cs_end: \__file_name_expand_end:
804   }

```

Now to the error case. `\__file_name_expand_error:Nw` adds an extra `\cs_end:` so that in case there was an extra `\csname` in the file name, then `\__file_name_expand_error_aux:Nw` throws the error.

```

805   \cs_new:Npn \__file_name_expand_error:Nw #1 #2 \__file_name_expand_end:
806   { \__file_name_expand_error_aux:Nw #1 #2 \cs_end: \__file_name_expand_end: }
807   \cs_new:Npn \__file_name_expand_error_aux:Nw #1 #2 \cs_end: #3
808     \__file_name_expand_end:
809   {
810     \msg_expandable_error:nnff
811       { kernel } { filename-chars-lost }
812       { \token_to_str:N #1 } { \exp_stop_f: #2 }
813   }

```

Quoting file name uses basically the same approach as for `luaquotejobname:` count the " tokens and remove them.

```

814   \cs_new:Npn \__file_name_strip_quotes:n #1
815   {
816     \__file_name_strip_quotes:nw { 0 }
817     #1 " \q__file_recursion_tail " \q__file_recursion_stop {#1}
818   }
819   \cs_new:Npn \__file_name_strip_quotes:nw #1#2 "
820   {
821     \if_meaning:w \q__file_recursion_tail #2
822       \__file_name_strip_quotes_end:w nwn

```

```

823     \fi:
824     #2
825     \__file_name_strip_quotes:nw { #1 + 1 }
826   }
827 \cs_new:Npn \__file_name_strip_quotes_end:wnwn \fi: #1
828   \__file_name_strip_quotes:nw #2 \q__file_recursion_stop #3
829   {
830     \fi:
831     \int_if_odd:nT {#2}
832     {
833       \msg_expandable_error:nnn
834         { kernel } { unbalanced-quote-in-filename } {#3}
835     }
836   }

```

Spaces need to be trimmed from the start of the name and from the end of any extension. However, the name we are passed might not have an extension: that means we have to look for one. If there is no extension, we still use the standard trimming function but deliberately prevent any spaces being removed at the end.

```

837 \cs_new:Npn \__file_name_trim_spaces:n #1
838   { \__file_name_trim_spaces:nw {#1} #1 . \q__file_nil . \s__file_stop }
839 \cs_new:Npn \__file_name_trim_spaces:nw #1#2 . #3 . #4 \s__file_stop
840   {
841     \__file_quark_if_nil:nTF {#3}
842     {
843       \tl_trim_spaces_apply:nN { #1 \s__file_stop }
844       \__file_name_trim_spaces_aux:n
845     }
846     { \tl_trim_spaces:n {#1} }
847   }
848 \cs_new:Npn \__file_name_trim_spaces_aux:n #1
849   { \__file_name_trim_spaces_aux:w #1 }
850 \cs_new:Npn \__file_name_trim_spaces_aux:w #1 \s__file_stop {#1}

```

(*\\_\_kernel\_file\_name\_sanitize:n* 以及其它的定义结束。)

```

__kernel_file_name_quote:n
  \__file_name_quote:nw 851 \cs_new:Npn \__kernel_file_name_quote:n #1
852   { \__file_name_quote:nw {#1} #1 ~ \q__file_nil \s__file_stop }
853 \cs_new:Npn \__file_name_quote:nw #1 #2 ~ #3 \s__file_stop
854   {
855     \__file_quark_if_nil:nTF {#3}
856     { #1 }

```

```

857     { "#1" }
858   }

```

(`\_kernel_file_name_quote:n` 和 `\_file_name_quote:nw` 定义结束。)

`\c__file_marker_tl` The same idea as the marker for rescanning token lists: this pair of tokens cannot appear in a file that is being input.

```

859 \tl_const:Ne \c__file_marker_tl { : \token_to_str:N : }

```

(`\c__file_marker_tl` 定义结束。)

`\file_get:nnN $\mathit{TF}$`  The approach here is similar to that for `\tl_set_rescan:Nnn`. The file contents are  
`\file_get:VnN $\mathit{TF}$`  grabbed as an argument delimited by `\c__file_marker_tl`. A few subtleties: braces  
`\file_get:nnN` in `\if_false: ... \fi:` to deal with possible alignment tabs, `\tracingnesting` to  
`\_file_get_aux:nnN` avoid a warning about a group being closed inside the `\scantokens`, and `\prg_-`  
`\_file_get_do:Nw` `return_true:` is placed after the end-of-file marker.

```

860 \cs_new_protected:Npn \file_get:nnN #1#2#3
861 {
862   \file_get:nnNF {#1} {#2} #3
863   { \tl_set:Nn #3 { \q_no_value } }
864 }
865 \cs_generate_variant:Nn \file_get:nnN { V }
866 \prg_new_protected_conditional:Npnn \file_get:nnN #1#2#3 { T , F , TF }
867 {
868   \file_get_full_name:nNTF {#1} \l__file_full_name_tl
869   {
870     \exp_args:NV \_file_get_aux:nnN
871       \l__file_full_name_tl
872       {#2} #3
873     \prg_return_true:
874   }
875   { \prg_return_false: }
876 }
877 \prg_generate_conditional_variant:Nnn \file_get:nnN { V } { T , F , TF }
878 \cs_new_protected:Npe \_file_get_aux:nnN #1#2#3
879 {
880   \exp_not:N \if_false: { \exp_not:N \fi:
881   \group_begin:
882     \int_set_eq:NN \tex_tracingnesting:D \c_zero_int
883     \exp_not:N \exp_args:No \tex_everyeof:D
884     { \exp_not:N \c__file_marker_tl }
885     #2 \scan_stop:

```

```

886 \exp_not:N \exp_after:wN \exp_not:N \_file_get_do:Nw
887 \exp_not:N \exp_after:wN #3
888 \exp_not:N \exp_after:wN \exp_not:N \prg_do_nothing:
889 \exp_not:N \tex_input:D
890 \sys_if_engine_luatex:TF
891 { {#1} }
892 { \exp_not:N \_kernel_file_name_quote:n {#1} \scan_stop: }
893 \exp_not:N \if_false: } \exp_not:N \fi:
894 }
895 \exp_args:Nno \use:nn
896 { \cs_new_protected:Npn \_file_get_do:Nw #1#2 }
897 { \c_file_marker_tl }
898 {
899 \group_end:
900 \tl_set:No #1 {#2}
901 }

```

(*\file\_get:nnNTF* 以及其它的定义结束。这些函数被记录在第20页。)

*\\_file\_size:n* A copy of the primitive where it's available.

```

902 \cs_new_eq:NN \_file_size:n \tex_filesize:D

```

(*\\_file\_size:n* 定义结束。)

*\file\_full\_name:n* File searching can be carried out if the *\pdffilesize* primitive or an equivalent is available. That of course means we need to arrange for everything else to here to be done by expansion too. We start off by sanitizing the name and quoting if required: *\\_file\_full\_name\_aux:n* we may need to remove those quotes, so the raw name is passed too.

```

\file_full_name_auxi:nn 903 \cs_new:Npn \file_full_name:n #1
\file_full_name_aux:Nnn 904 {
\file_full_name_slash:n 905 \exp_args:Ne \_file_full_name:n
\file_full_name_slash:w 906 { \_kernel_file_name_sanitiz:n {#1} }
907 }
\file_full_name_aux:nN 908 \cs_generate_variant:Nn \file_full_name:n { V }
\file_full_name_aux:nnN

```

First, we check of the file is just here: no mapping so we do not need the break part of the broader auxiliary. We are using the fact that the primitive here returns nothing if the file is entirely absent. To avoid unnecessary filesystem lookups, the result of *\pdffilesize* is kept available as an argument. For package mode, *\input@path* is a token list not a sequence.

```

\file_name_cleanup:w
\file_name_end:
\file_name_ext_check:nn 909 \cs_new:Npn \_file_full_name:n #1
\file_name_ext_check:nnw 910 {
\file_name_ext_check:nnnw
\file_name_ext_check:nnnn

```



```

911     \tl_if_blank:nF {#1}
912     { \exp_args:Nne \__file_full_name_auxii:nn {#1} { \__file_full_name_aux:n {#1} } }
913 }

```

To avoid repeated reading of files we need to cache the loading: this is important as the code here is used by *all* file checks. The same marker is used in the  $\text{\LaTeX} 2_{\epsilon}$  kernel, meaning that we get a double-saving with for example `\IfFileExists`. As this is all about performance, we use the low-level approach for the conditionals. For a file already seen, the size is reported as `-1` so it's distinct from any non-cached ones.

```

914 \cs_new:Npn \__file_full_name_aux:n #1
915 {
916     \if_cs_exist:w __file_seen_ \tl_to_str:n {#1} : \cs_end:
917     -1
918     \else:
919         \exp_args:Ne \__file_full_name_auxi:nn { \__file_size:n {#1} } {#1}
920     \fi:
921 }

```

We will need the size of files later, and we have to avoid the `\scan_stop:` causing issues if we are raising the flag. Thus there is a slightly odd gobble here.

```

922 \cs_new:Npn \__file_full_name_auxi:nn #1#2
923 {
924     \if:w \scan_stop: #1 \scan_stop:
925     \else:
926         \exp_after:wN \use_none:n
927         \cs:w __file_seen_ \tl_to_str:n {#2} : \cs_end:
928         #1
929     \fi:
930 }
931 \cs_new:Npn \__file_full_name_auxii:nn #1 #2
932 {
933     \tl_if_blank:nTF {#2}
934     {
935         \seq_map_tokens:Nn \l_file_search_path_seq
936         { \__file_full_name_aux:Nnn \seq_map_break:n {#1} }
937         \cs_if_exist:NT \input@path
938         {
939             \tl_map_tokens:Nn \input@path
940             { \__file_full_name_aux:Nnn \tl_map_break:n {#1} }
941         }
942         \__file_name_end:

```

```

943     }
944     { \_file_ext_check:nn {#1} {#2} }
945 }

```

Two pars to the auxiliary here so we can avoid doing quoting twice in the event we find the right file.

```

946 \cs_new:Npn \_file_full_name_aux:Nnn #1#2#3
947 {
948   \exp_args:Ne \_file_full_name_aux:nN
949   { \_file_full_name_slash:n {#3} #2 }
950   #1
951 }
952 \cs_new:Npn \_file_full_name_slash:n #1
953 {
954   \_file_full_name_slash:nw {#1} #1 \q_nil / \q_nil / \q_nil \q_stop
955 }
956 \cs_new:Npn \_file_full_name_slash:nw #1#2 / \q_nil / #3 \q_stop
957 {
958   \quark_if_nil:nTF {#3}
959     { #1 / }
960     { #2 / }
961 }
962 \cs_new:Npn \_file_full_name_aux:nN #1
963 { \exp_args:Nne \_file_full_name_aux:nnN {#1} { \_file_full_name_aux:n {#1} } }
964 \cs_new:Npn \_file_full_name_aux:nnN #1 #2 #3
965 {
966   \tl_if_blank:nF {#2}
967   {
968     #3
969     {
970       \_file_ext_check:nn {#1} {#2}
971       \_file_name_cleanup:w
972     }
973   }
974 }
975 \cs_new:Npn \_file_name_cleanup:w #1 \_file_name_end: { }
976 \cs_new:Npn \_file_name_end: { }

```

As  $\text{\TeX}$  automatically adds `.tex` if there is no extension, there is a little clean up to do here. First, make sure we are not in the directory part, saving that. Then check for an extension.

```

977 \cs_new:Npn \_file_ext_check:nn #1 #2
978 { \_file_ext_check:nnw {#2} { / } #1 / \q_file_nil / \s_file_stop }

```

```

979 \cs_new:Npn \__file_ext_check:nnw #1 #2 #3 / #4 / #5 \s__file_stop
980 {
981   \__file_quark_if_nil:nTF {#4}
982   {
983     \exp_args:No \__file_ext_check:nnnw
984     { \use_none:n #2 } {#1} {#3} #3 . \q__file_nil . \s__file_stop
985   }
986   { \__file_ext_check:nnw {#1} { #2 #3 / } #4 / #5 \s__file_stop }
987 }
988 \cs_new:Npe \__file_ext_check:nnnw #1#2#3#4 . #5 . #6 \s__file_stop
989 {
990   \exp_not:N \__file_quark_if_nil:nTF {#5}
991   {
992     \exp_not:N \__file_ext_check:nnn
993     { #1 #3 \tl_to_str:n { .tex } } { #1 #3 } {#2}
994   }
995   { #1 #3 }
996 }
997 \cs_new:Npn \__file_ext_check:nnn #1
998 { \exp_args:Nne \__file_ext_check:nnnn {#1} { \__file_full_name_aux:n {#1} } }
999 \cs_new:Npn \__file_ext_check:nnnn #1#2#3#4
1000 {
1001   \tl_if_blank:nTF {#2}
1002   {#3}
1003   {
1004     \bool_lazy_or:nnTF
1005     { \int_compare_p:nNn {#4} = {#2} }
1006     { \int_compare_p:nNn {#2} = { -1 } }
1007     {#1}
1008     {#3}
1009   }
1010 }

```

(`\file_full_name:n` 以及其它的定义结束。这个函数被记录在第18页。)

`\file_get_full_name:nN` These functions pre-date using `\tex_filesize:D` for file searching, so are `get` functions with protection. To avoid having different search set ups, they are simply wrappers around the code above.

`\file_get_full_name:VN`

`\file_get_full_name:nN $\underline{TF}$`

`\file_get_full_name:VN $\underline{TF}$`

`\__file_get_full_name_search:nN`

```

1011 \cs_new_protected:Npn \file_get_full_name:nN #1#2
1012 {
1013   \file_get_full_name:nNF {#1} #2
1014   { \tl_set:Nn #2 { \q_no_value } }

```

```

1015     }
1016 \cs_generate_variant:Nn \file_get_full_name:nN { V }
1017 \prg_new_protected_conditional:Npnn \file_get_full_name:nN #1#2 { T , F , TF }
1018 {
1019     \__kernel_tl_set:Ne #2
1020     { \file_full_name:n {#1} }
1021     \tl_if_empty:NTF #2
1022     { \prg_return_false: }
1023     { \prg_return_true: }
1024 }
1025 \prg_generate_conditional_variant:Nnn \file_get_full_name:nN
1026 { V } { T , F , TF }

```

(`\file_get_full_name:nN`, `\file_get_full_name:nNTF`, 和 `\__file_get_full_name_search:nN` 定义结束。这些函数被记录在第18页。)

`\g__file_internal_ior` A reserved stream to test for opening a shell.

```

1027 \ior_new:N \g__file_internal_ior

```

(`\g__file_internal_ior` 定义结束。)

`\file_md5five_hash:n` Getting file details by expansion is relatively easy if a bit repetitive. As the MD5 function has a slightly different syntax from the other commands, there is a little cleaning up to do.

`\file_md5five_hash:V`

`\file_size:n`

`\file_size:V`

`\file_timestamp:n`

`\file_timestamp:V`

`\__file_details:nn`

`\__file_details_aux:nn`

`\__file_md5five_hash:n`

```

1028 \cs_new:Npn \file_size:n #1
1029 { \__file_details:nn {#1} { size } }
1030 \cs_generate_variant:Nn \file_size:n { V }
1031 \cs_new:Npn \file_timestamp:n #1
1032 { \__file_details:nn {#1} { moddate } }
1033 \cs_generate_variant:Nn \file_timestamp:n { V }
1034 \cs_new:Npn \__file_details:nn #1#2
1035 {
1036     \exp_args:Ne \__file_details_aux:nn
1037     { \file_full_name:n {#1} } {#2}
1038 }
1039 \cs_new:Npn \__file_details_aux:nn #1#2
1040 {
1041     \tl_if_blank:nF {#1}
1042     { \use:c { tex_file #2 :D } {#1} }
1043 }
1044 \cs_new:Npn \file_md5five_hash:n #1
1045 { \exp_args:Ne \__file_md5five_hash:n { \file_full_name:n {#1} } }
1046 \cs_generate_variant:Nn \file_md5five_hash:n { V }

```

```

1047 \cs_new:Npn \__file_mdfive_hash:n #1
1048 { \tex_mdffivesum:D file {#1} }

```

(*\file\_mdfive\_hash:n* 以及其它的定义结束。这些函数被记录在第17页。)

*\file\_hex\_dump:nnn* These are separate as they need multiple arguments *or* the file size. For LuaTeX, the emulation does not need the file size so we save a little on expansion.

*\file\_hex\_dump:Vnn*

*\\_\_file\_hex\_dump\_auxi:nnn*

```

1049 \cs_new:Npn \file_hex_dump:nnn #1#2#3

```

```

1050 {

```

```

1051   \exp_args:Neee \__file_hex_dump_auxi:nnn

```

```

1052     { \file_full_name:n {#1} }

```

```

1053     { \int_eval:n {#2} }

```

```

1054     { \int_eval:n {#3} }

```

```

1055 }

```

*\\_\_file\_hex\_dump:n*

```

1056 \cs_generate_variant:Nn \file_hex_dump:nnn { V }

```

```

1057 \cs_new:Npn \__file_hex_dump_auxi:nnn #1#2#3

```

```

1058 {

```

```

1059   \bool_lazy_any:nF

```

```

1060   {

```

```

1061     { \tl_if_blank_p:n {#1} }

```

```

1062     { \int_compare_p:nNn {#2} = 0 }

```

```

1063     { \int_compare_p:nNn {#3} = 0 }

```

```

1064   }

```

```

1065   {

```

```

1066     \exp_args:Ne \__file_hex_dump_auxii:nnnn

```

```

1067     { \__file_details_aux:nn {#1} { size } }

```

```

1068     {#1} {#2} {#3}

```

```

1069   }

```

```

1070 }

```

```

1071 \cs_new:Npn \__file_hex_dump_auxii:nnnn #1#2#3#4

```

```

1072 {

```

```

1073   \int_compare:nNnTF {#3} > 0

```

```

1074     { \__file_hex_dump_auxiii:nnnn {#3} }

```

```

1075     {

```

```

1076       \exp_args:Ne \__file_hex_dump_auxiii:nnnn

```

```

1077       { \int_eval:n { #1 + #3 } }

```

```

1078     }

```

```

1079     {#1} {#2} {#4}

```

```

1080 }

```

```

1081 \cs_new:Npn \__file_hex_dump_auxiii:nnnn #1#2#3#4

```

```

1082 {

```

```

1083   \int_compare:nNnTF {#4} > 0

```

```

1084     { \__file_hex_dump_auxiv:nnn {#4} }

```

```

1085     {
1086       \exp_args:Ne \__file_hex_dump_auxiv:nnn
1087       { \int_eval:n { #2 + #4 } }
1088     }
1089     {#1} {#3}
1090   }
1091   \cs_new:Npn \__file_hex_dump_auxiv:nnn #1#2#3
1092   {
1093     \tex_filedump:D
1094     offset ~ \int_eval:n { #2 - 1 } ~
1095     length ~ \int_eval:n { #1 - #2 + 1 }
1096     {#3}
1097   }
1098   \cs_new:Npn \file_hex_dump:n #1
1099   { \exp_args:Ne \__file_hex_dump:n { \file_full_name:n {#1} } }
1100   \cs_generate_variant:Nn \file_hex_dump:n { V }
1101   \sys_if_engine luatex:TF
1102   {
1103     \cs_new:Npn \__file_hex_dump:n #1
1104     {
1105       \tl_if_blank:nF {#1}
1106       { \tex_filedump:D whole {#1} {#1} }
1107     }
1108   }
1109   {
1110     \cs_new:Npn \__file_hex_dump:n #1
1111     {
1112       \tl_if_blank:nF {#1}
1113       { \tex_filedump:D length \tex_filesize:D {#1} {#1} }
1114     }
1115   }

```

(*\file\_hex\_dump:nnn* 以及其它的定义结束。这些函数被记录在第16页。)

<code>\file_get_hex_dump:nN</code> <code>\file_get_hex_dump:VN</code> <code>\file_get_hex_dump:nNTF</code> <code>\file_get_hex_dump:VNTF</code> <code>\file_get_md5five_hash:nN</code> <code>\file_get_md5five_hash:VN</code> <code>\file_get_md5five_hash:nNTF</code> <code>\file_get_md5five_hash:VNTF</code> <code>\file_get_size:nN</code> <code>\file_get_size:VN</code> <code>\file_get_size:nNTF</code> <code>\file_get_size:VNTF</code> <code>\file_get_timestamp:nN</code> <code>\file_get_timestamp:VN</code> <code>\file_get_timestamp:nNTF</code>	Non-expandable wrappers around the above in the case where appropriate primitive support exists.  <code>\cs_new_protected:Npn \file_get_hex_dump:nN #1#2</code> <code>{ \file_get_hex_dump:nNF {#1} #2 { \tl_set:Nn #2 { \q_no_value } } }</code> <code>\cs_generate_variant:Nn \file_get_hex_dump:nN { V }</code> <code>\cs_new_protected:Npn \file_get_md5five_hash:nN #1#2</code> <code>{ \file_get_md5five_hash:nNF {#1} #2 { \tl_set:Nn #2 { \q_no_value } } }</code> <code>\cs_generate_variant:Nn \file_get_md5five_hash:nN { V }</code> <code>\cs_new_protected:Npn \file_get_size:nN #1#2</code>
---	---

```

1123 { \file_get_size:nNF {#1} #2 { \tl_set:Nn #2 { \q_no_value } } }
1124 \cs_generate_variant:Nn \file_get_size:nN { V }
1125 \cs_new_protected:Npn \file_get_timestamp:nN #1#2
1126 { \file_get_timestamp:nNF {#1} #2 { \tl_set:Nn #2 { \q_no_value } } }
1127 \cs_generate_variant:Nn \file_get_timestamp:nN { V }
1128 \prg_new_protected_conditional:Npnn \file_get_hex_dump:nN #1#2 { T , F , TF }
1129 { \__file_get_details:nnN {#1} { hex_dump } #2 }
1130 \prg_generate_conditional_variant:Nnn \file_get_hex_dump:nN
1131 { V } { T , F , TF }
1132 \prg_new_protected_conditional:Npnn \file_get_md5five_hash:nN #1#2 { T , F , TF }
1133 { \__file_get_details:nnN {#1} { md5five_hash } #2 }
1134 \prg_generate_conditional_variant:Nnn \file_get_md5five_hash:nN
1135 { V } { T , F , TF }
1136 \prg_new_protected_conditional:Npnn \file_get_size:nN #1#2 { T , F , TF }
1137 { \__file_get_details:nnN {#1} { size } #2 }
1138 \prg_generate_conditional_variant:Nnn \file_get_size:nN
1139 { V } { T , F , TF }
1140 \prg_new_protected_conditional:Npnn \file_get_timestamp:nN #1#2 { T , F , TF }
1141 { \__file_get_details:nnN {#1} { timestamp } #2 }
1142 \prg_generate_conditional_variant:Nnn \file_get_timestamp:nN
1143 { V } { T , F , TF }
1144 \cs_new_protected:Npn \__file_get_details:nnN #1#2#3
1145 {
1146   \__kernel_tl_set:Ne #3
1147   { \use:c { file_ #2 :n } {#1} }
1148   \tl_if_empty:NTF #3
1149   { \prg_return_false: }
1150   { \prg_return_true: }
1151 }

```

(*\file\_get\_hex\_dump:nNTF* 以及其它的定义结束。这些函数被记录在第16页。)

```

\file_get_hex_dump:nnnN Custom code due to the additional arguments.
\file_get_hex_dump:VnnN 1152 \cs_new_protected:Npn \file_get_hex_dump:nnnN #1#2#3#4
\file_get_hex_dump:nnnNTF 1153 {
\file_get_hex_dump:VnnNTF 1154   \file_get_hex_dump:nnnNF {#1} {#2} {#3} #4
1155   { \tl_set:Nn #4 { \q_no_value } }
1156 }
1157 \cs_generate_variant:Nn \file_get_hex_dump:nnnN { V }
1158 \prg_new_protected_conditional:Npnn \file_get_hex_dump:nnnN #1#2#3#4
1159 { T , F , TF }
1160 {
1161   \__kernel_tl_set:Ne #4

```

```

1162     { \file_hex_dump:nnn {#1} {#2} {#3} }
1163     \tl_if_empty:NTF #4
1164     { \prg_return_false: }
1165     { \prg_return_true: }
1166   }
1167   \prg_generate_conditional_variant:Nnn \file_get_hex_dump:nnnN
1168   { V } { T , F , TF }

```

(*\file\_get\_hex\_dump:nnnNTF* 定义结束。这个函数被记录在第16页。)

**\\_file\_str\_cmp:nn** As we are doing a fixed-length “big” integer comparison, it is easiest to use the low-level behavior of string comparisons.

```

1169 \cs_new_eq:NN \_file_str_cmp:nn \tex_strcmp:D

```

(*\\_file\_str\_cmp:nn* 定义结束。)

**\file\_compare\_timestamp:p:nN** Comparison of file date can be done by using the low-level nature of the string comparison functions.

```

\file_compare_timestamp:p:nN 1170 \prg_new_conditional:Npnn \file_compare_timestamp:nNn #1#2#3
\file_compare_timestamp:p:nNV 1171 { p , T , F , TF }
\file_compare_timestamp:p:VNn 1172 {
\file_compare_timestamp:nNnTF 1173   \exp_args:Nee \_file_compare_timestamp:nnN
\file_compare_timestamp:nNVTF 1174   { \file_full_name:n {#1} }
\file_compare_timestamp:VNnTF 1175   { \file_full_name:n {#3} }
\file_compare_timestamp:VNVTF 1176   #2
\_file_compare_timestamp:nnN 1177 }
\_file_timestamp:n 1178 \prg_generate_conditional_variant:Nnn \file_compare_timestamp:nNn
1179 { nNV , V , VNV } { p , T , F , TF }
1180 \cs_new:Npn \_file_compare_timestamp:nnN #1#2#3
1181 {
1182   \tl_if_blank:nTF {#1}
1183   {
1184     \if_charcode:w #3 <
1185     \prg_return_true:
1186     \else:
1187     \prg_return_false:
1188     \fi:
1189   }
1190   {
1191     \tl_if_blank:nTF {#2}
1192     {
1193       \if_charcode:w #3 >
1194       \prg_return_true:

```



```

1195         \else:
1196         \prg_return_false:
1197         \fi:
1198     }
1199     {
1200         \if_int_compare:w
1201             \__file_str_cmp:nn
1202             { \__file_timestamp:n {#1} }
1203             { \__file_timestamp:n {#2} }
1204             #3 \c_zero_int
1205         \prg_return_true:
1206         \else:
1207         \prg_return_false:
1208         \fi:
1209     }
1210 }
1211 }
1212 \cs_new_eq:NN \__file_timestamp:n \tex_filemoddate:D

```

(`\file_compare_timestamp:nNnTF`, `\__file_compare_timestamp:nnN`, 和 `\__file_timestamp:n` 定义结束。这个函数被记录在第18页。)

`\file_if_exist_p:n` The test for the existence of a file is a wrapper around the function to add a path to a file. If the file was found, the path contains something, whereas if the file was not located then the return value is empty.

`\file_if_exist_p:V`

`\file_if_exist:nTF`

`\file_if_exist:VTF`

```

1213 \prg_new_conditional:Npnn \file_if_exist:n #1 { p , T , F , TF }
1214 {
1215     \tl_if_blank:eTF { \file_full_name:n {#1} }
1216     { \prg_return_false: }
1217     { \prg_return_true: }
1218 }
1219 \prg_generate_conditional_variant:Nnn \file_if_exist:n { V } { p , T , F , TF }

```

(`\file_if_exist:nTF` 定义结束。这个函数被记录在第16页。)

`\file_if_exist_input:n` Input of a file with a test for existence. We do not define the T or TF variants because the most useful place to place the *<true code>* would be inconsistent with other conditionals.

`\file_if_exist_input:V`

`\file_if_exist_input:nF`

`\file_if_exist_input:VF`

```

1220 \cs_new_protected:Npn \file_if_exist_input:n #1
1221 {
1222     \file_get_full_name:nNT {#1} \l__file_full_name_tl
1223     { \__file_input:V \l__file_full_name_tl }
1224 }

```

```

1225 \cs_generate_variant:Nn \file_if_exist_input:n { V }
1226 \cs_new_protected:Npn \file_if_exist_input:nF #1#2
1227 {
1228     \file_get_full_name:nNTF {#1} \l__file_full_name_tl
1229     { \__file_input:V \l__file_full_name_tl }
1230     {#2}
1231 }
1232 \cs_generate_variant:Nn \file_if_exist_input:nF { V }

(\file_if_exist_input:n 和 \file_if_exist_input:nF 定义结束。这些函数被记录在第20页。)

```

`\file_input_stop:` A simple rename.

```

1233 \cs_new_protected:Npn \file_input_stop: { \tex_endinput:D }

(\file_input_stop: 定义结束。这个函数被记录在第21页。)

```

`\__kernel_file_missing:n` An error message for a missing file, also used in `\ior_open:Nn`.

```

1234 \cs_new_protected:Npn \__kernel_file_missing:n #1
1235 {
1236     \msg_error:nne { kernel } { file-not-found }
1237     { \__kernel_file_name_sanitiz:n {#1} }
1238 }

(\__kernel_file_missing:n 定义结束。)

```

`\file_input:n` Loading a file is done in a safe way, checking first that the file exists and loading  
`\file_input:V` only if it does. Push the file name on the `\g__file_stack_seq`, and add it to the  
`\__file_input:n` file list, either `\g__file_record_seq`, or `\@filelist` in package mode.

```

\__file_input:V 1239 \cs_new_protected:Npn \file_input:n #1
\__file_input_push:n 1240 {
\__kernel_file_input_push:n 1241     \file_get_full_name:nNTF {#1} \l__file_full_name_tl
\__file_input_pop: 1242     { \__file_input:V \l__file_full_name_tl }
\__kernel_file_input_pop: 1243     { \__kernel_file_missing:n {#1} }
\__file_input_pop:nnn 1244 }
1245 \cs_generate_variant:Nn \file_input:n { V }
1246 \cs_new_protected:Npe \__file_input:n #1
1247 {
1248     \exp_not:N \clist_if_exist:NTF \exp_not:N \@filelist
1249     { \exp_not:N \@addtofilelist {#1} }
1250     { \seq_gput_right:Nn \exp_not:N \g__file_record_seq {#1} }
1251     \exp_not:N \__file_input_push:n {#1}
1252     \exp_not:N \tex_input:D
1253     \sys_if_engine_luatex:TF

```

```

1254     { {#1} }
1255     { \exp_not:N \__kernel_file_name_quote:n {#1} \scan_stop: }
1256     \exp_not:N \__file_input_pop:
1257   }
1258   \cs_generate_variant:Nn \__file_input:n { V }

```

Keeping a track of the file data is easy enough: we store the separated parts so we do not need to parse them twice.

```

1259   \cs_new_protected:Npn \__file_input_push:n #1
1260   {
1261     \seq_gpush:Ne \g__file_stack_seq
1262     {
1263       { \g_file_curr_dir_str }
1264       { \g_file_curr_name_str }
1265       { \g_file_curr_ext_str }
1266     }
1267     \file_parse_full_name:nNNN {#1}
1268     \l__file_dir_str \l__file_name_str \l__file_ext_str
1269     \str_gset_eq:NN \g_file_curr_dir_str \l__file_dir_str
1270     \str_gset_eq:NN \g_file_curr_name_str \l__file_name_str
1271     \str_gset_eq:NN \g_file_curr_ext_str \l__file_ext_str
1272   }
1273   \cs_new_eq:NN \__kernel_file_input_push:n \__file_input_push:n
1274   \cs_new_protected:Npn \__file_input_pop:
1275   {
1276     \seq_gpop:NN \g__file_stack_seq \l__file_internal_tl
1277     \exp_after:wN \__file_input_pop:nnn \l__file_internal_tl
1278   }
1279   \cs_new_eq:NN \__kernel_file_input_pop: \__file_input_pop:
1280   \cs_new_protected:Npn \__file_input_pop:nnn #1#2#3
1281   {
1282     \str_gset:Nn \g_file_curr_dir_str {#1}
1283     \str_gset:Nn \g_file_curr_name_str {#2}
1284     \str_gset:Nn \g_file_curr_ext_str {#3}
1285   }

```

(`\file_input:n` 以及其它的定义结束。这个函数被记录在第20页。)

```

\file_input_raw:n No error checking, no tracking.
\file_input_raw:V
\__file_input_raw:nn
1286   \cs_new:Npn \file_input_raw:n #1
1287   { \exp_args:Ne \__file_input_raw:nn { \file_full_name:n {#1} } {#1} }
1288   \cs_generate_variant:Nn \file_input_raw:n { V }
1289   \cs_new:Npe \__file_input_raw:nn #1#2

```

```

1290 {
1291   \exp_not:N \tl_if_blank:nTF {#1}
1292   {
1293     \exp_not:N \exp_args:Nne \exp_not:N \msg_expandable_error:nnn
1294     { kernel } { file-not-found }
1295     { \exp_not:N \__kernel_file_name_sanitiz:n {#2} }
1296   }
1297   {
1298     \exp_not:N \tex_input:D
1299     \sys_if_engine luatex:TF
1300     { {#1} }
1301     { \exp_not:N \__kernel_file_name_quote:n {#1} \scan_stop: }
1302   }
1303 }
1304 \exp_args_generate:n { nne }

```

(*\file\_input\_raw:n* 和 *\\_\_file\_input\_raw:nn* 定义结束。这个函数被记录在第20页。)

*\file\_parse\_full\_name:n* The main parsing macro *\file\_parse\_full\_name\_apply:nN* passes the file name #1 through *\\_\_kernel\_file\_name\_sanitiz:n* so that we have a single normalised way to treat files internally. *\file\_parse\_full\_name:n* uses the former, with *\prg\_do\_nothing:* to leave each part of the name within a pair of braces.

```

1305 \cs_new:Npn \file_parse_full_name:n #1
1306 {
1307   \file_parse_full_name_apply:nN {#1}
1308   \prg_do_nothing:
1309 }
1310 \cs_generate_variant:Nn \file_parse_full_name:n { V }
1311 \cs_new:Npn \file_parse_full_name_apply:nN #1
1312 {
1313   \exp_args:Ne \__file_parse_full_name_auxi:nN
1314   { \__kernel_file_name_sanitiz:n {#1} }
1315 }
1316 \cs_generate_variant:Nn \file_parse_full_name_apply:nN { V }

```

*\\_\_file\_parse\_full\_name\_area:nw* splits the file name into chunks separated by /, until the last one is reached. The last chunk is the file name plus the extension, and everything before that is the path. When *\\_\_file\_parse\_full\_name\_area:nw* is done, it leaves the path within braces after the scan mark *\s\_\_file\_stop* and proceeds parsing the actual file name.

```

\__file_parse_full_name_auxi:nN
\__file_parse_full_name_area:nw 1317 \cs_new:Npn \__file_parse_full_name_auxi:nN #1

```

```

1318 {
1319     \_file_parse_full_name_area:nw { } #1
1320     / \s__file_stop
1321 }
1322 \cs_new:Npn \_file_parse_full_name_area:nw #1 #2 / #3 \s__file_stop
1323 {
1324     \tl_if_empty:nTF {#3}
1325     { \_file_parse_full_name_base:nw { } #2 . \s__file_stop {#1} }
1326     { \_file_parse_full_name_area:nw { #1 / #2 } #3 \s__file_stop }
1327 }

```

\\_file\_parse\_full\_name\_base:nw does roughly the same as above, but it separates the chunks at each period. However here there's some extra complications: In case #1 is empty, it is assumed that the extension is actually empty, and the file name is #2. Besides, an extra . has to be added to #2 because it is later removed in \\_file\_parse\_full\_name\_tidy:nnnN. In any case, if there's an extension, it is returned with a leading ..

```

\_file_parse_full_name_base:nw 1328 \cs_new:Npn \_file_parse_full_name_base:nw #1 #2 . #3 \s__file_stop
1329 {
1330     \tl_if_empty:nTF {#3}
1331     {
1332         \tl_if_empty:nTF {#1}
1333         {
1334             \tl_if_empty:nTF {#2}
1335             { \_file_parse_full_name_tidy:nnnN { } { } }
1336             { \_file_parse_full_name_tidy:nnnN { .#2 } { } }
1337         }
1338         { \_file_parse_full_name_tidy:nnnN {#1} { .#2 } }
1339     }
1340     { \_file_parse_full_name_base:nw { #1 . #2 } #3 \s__file_stop }
1341 }

```

Now we just need to tidy some bits left loose before. The loop used in the two macros above start with a leading / and . in the file path an name, so here we need to remove them, except in the path, if it is a single /, in which case it's left as is. After all's done, pass to #4.

```

1342 \cs_new:Npn \_file_parse_full_name_tidy:nnnN #1 #2 #3 #4
1343 {
1344     \exp_args:Nee #4
1345     {
1346         \str_if_eq:nnF {#3} { / } { \use_none:n }

```

```

1347         #3 \prg_do_nothing:
1348     }
1349     { \use_none:n #1 \prg_do_nothing: }
1350     {#2}
1351 }

```

(\file\_parse\_full\_name:n 以及其它的定义结束。这些函数被记录在第19页。)

\file\_parse\_full\_name:nNNN

\file\_parse\_full\_name:VNNN

```

1352 \cs_new_protected:Npn \file_parse_full_name:nNNN #1 #2 #3 #4
1353 {
1354     \file_parse_full_name_apply:nN {#1}
1355     \__file_full_name_assign:nnnNNN #2 #3 #4
1356 }
1357 \cs_new_protected:Npn \__file_full_name_assign:nnnNNN #1 #2 #3 #4 #5 #6
1358 {
1359     \str_set:Nn #4 {#1}
1360     \str_set:Nn #5 {#2}
1361     \str_set:Nn #6 {#3}
1362 }
1363 \cs_generate_variant:Nn \file_parse_full_name:nNNN { V }

```

(\file\_parse\_full\_name:nnnn 定义结束。这个函数被记录在第19页。)

\file\_show\_list: A function to list all files used to the log, without duplicates. In package mode, if  
\file\_log\_list: \@filelist is still defined, we need to take this list of file names into account (we  
\\_\_file\_list:N capture it \AtBeginDocument into \g\_\_file\_record\_seq), turning it to a string  
\\_\_file\_list\_aux:n (this does not affect the commas of this comma list).

```

1364 \cs_new_protected:Npn \file_show_list: { \__file_list:N \msg_show:nneeee }
1365 \cs_new_protected:Npn \file_log_list: { \__file_list:N \msg_log:nneeee }
1366 \cs_new_protected:Npn \__file_list:N #1
1367 {
1368     \seq_clear:N \l__file_tmp_seq
1369     \clist_if_exist:NT \@filelist
1370     {
1371         \exp_args:NNe \seq_set_from_clist:Nn \l__file_tmp_seq
1372             { \tl_to_str:N \@filelist }
1373     }
1374     \seq_concat:NNN \l__file_tmp_seq \l__file_tmp_seq \g__file_record_seq
1375     \seq_remove_duplicates:N \l__file_tmp_seq
1376     #1 { kernel } { file-list }
1377     { \seq_map_function:NN \l__file_tmp_seq \__file_list_aux:n }
1378     { } { } { }

```

```

1379 }
1380 \cs_new:Npn \__file_list_aux:n #1 { \iow_newline: #1 }

```

(*\file\_show\_list:* 以及其它的定义结束。这些函数被记录在第21页。)

When used as a package, there is a need to hold onto the standard file list as well as the new one here. File names recorded in `\@filelist` must be turned to strings before being added to `\g__file_record_seq`.

```

1381 \cs_if_exist:NT \@filelist
1382 {
1383   \AtBeginDocument
1384   {
1385     \exp_args:NNe \seq_set_from_clist:Nn \l__file_tmp_seq
1386     { \tl_to_str:N \@filelist }
1387     \seq_gconcat:NNN
1388     \g__file_record_seq
1389     \g__file_record_seq
1390     \l__file_tmp_seq
1391   }
1392 }

```

### 3.5 GetIdInfo

`\GetIdInfo` As documented in `expl3.dtx` this function extracts file name etc from an SVN Id line.

`\__file_id_info_auxi:w` This used to be how we got version number and so on in all modules, so it had to

`\__file_id_info_auxii:w` be defined in `l3bootstrap`. Now it's more convenient to define it after we have set up

`\__file_id_info_auxiii:w` quite a lot of tools, and `l3file` seems the least unreasonable place for it.

The idea here is to extract out the information needed from a standard SVN Id line, but to avoid a line that would get changed when the file is checked in. Hence the fact that none of the lines here include both a dollar sign and the Id keyword!

```

1393 \cs_new_protected:Npn \GetIdInfo
1394 {
1395   \tl_clear_new:N \ExplFileDescription
1396   \tl_clear_new:N \ExplFileDate
1397   \tl_clear_new:N \ExplFileName
1398   \tl_clear_new:N \ExplFileExtension
1399   \tl_clear_new:N \ExplFileVersion
1400   \group_begin:
1401   \char_set_catcode_space:n { 32 }
1402   \exp_after:wN
1403   \group_end:
1404   \__file_id_info_auxi:w

```

```
1405 }
```

A first check for a completely empty SVN field. If that is not the case, there is a second case when a file created using `svn cp` but has not been checked in. That leaves a special marker `-1` version, which has no further data. Dealing correctly with that is the reason for the space in the line to use `\_file_id_info_auxii:w`.

```
1406 \cs_new_protected:Npn \_file_id_info_auxii:w $ #1 $ #2
1407 {
1408   \tl_set:Nn \ExplFileDescription {#2}
1409   \str_if_eq:nnTF {#1} { Id }
1410   {
1411     \tl_set:Nn \ExplFileDate { 0000/00/00 }
1412     \tl_set:Nn \ExplFileName { [unknown] }
1413     \tl_set:Nn \ExplFileExtension { [unknown~extension] }
1414     \tl_set:Nn \ExplFileVersion {-1}
1415   }
1416   { \_file_id_info_auxii:w #1 ~ \s__file_stop }
1417 }
```

Here, `#1` is `Id`, `#2` is the file name, `#3` is the extension, `#4` is the version, `#5` is the check in date and `#6` is the check in time and user, plus some trailing spaces. If `#4` is the marker `-1` value then `#5` and `#6` are empty.

```
1418 \cs_new_protected:Npn \_file_id_info_auxii:w
1419   #1 ~ #2.#3 ~ #4 ~ #5 ~ #6 \s__file_stop
1420 {
1421   \tl_set:Nn \ExplFileName {#2}
1422   \tl_set:Nn \ExplFileExtension {#3}
1423   \tl_set:Nn \ExplFileVersion {#4}
1424   \str_if_eq:nnTF {#4} {-1}
1425   { \tl_set:Nn \ExplFileDate { 0000/00/00 } }
1426   { \_file_id_info_auxiii:w #5 - 0 - 0 - \s__file_stop }
1427 }
```

Convert an SVN-style date into a  $\text{\LaTeX}$ -style one.

```
1428 \cs_new_protected:Npn \_file_id_info_auxiii:w #1 - #2 - #3 - #4 \s__file_stop
1429 { \tl_set:Nn \ExplFileDate { #1/#2/#3 } }
```

(`\GetIdInfo` 以及其它的定义结束。这个函数被记录在第??页。)

### 3.6 Checking the version of kernel dependencies

`\_kernel_dependency_version_check:Nn` This function is responsible for checking if dependencies of the  $\text{\LaTeX}$ 3 kernel match the version preloaded in the  $\text{\LaTeX}$ 2<sub>ε</sub> kernel. If versions don't match, the function

`\_kernel_dependency_version_check:nn`

`\_file_kernel_dependency_compare:nnn`

`\_file_parse_version:w`



attempts to tell why by searching for a possible stray format file.

The function starts by checking that the kernel date is defined, and if not zero is used to force the error route. The kernel date is then compared with the argument requested date (usually the packaging date of the dependency). If the kernel date is less than the required date, it's an error and the loading should abort.

```

1430 \cs_new_protected:Npn \__kernel_dependency_version_check:Nn #1
1431 { \exp_args:NV \__kernel_dependency_version_check:nn #1 }
1432 \cs_new_protected:Npn \__kernel_dependency_version_check:nn #1
1433 {
1434   \cs_if_exist:NTF \c__kernel_expl_date_tl
1435   {
1436     \exp_args:NV \__file_kernel_dependency_compare:nnn
1437     \c__kernel_expl_date_tl {#1}
1438   }
1439   { \__file_kernel_dependency_compare:nnn { 0000-00-00 } {#1} }
1440 }
1441 \cs_new_protected:Npn \__file_kernel_dependency_compare:nnn #1 #2 #3
1442 {
1443   \int_compare:nNnT
1444   { \__file_parse_version:w #1 \s__file_stop } <
1445   { \__file_parse_version:w #2 \s__file_stop }
1446   { \__file_mismatched_dependency_error:nn {#2} {#3} }
1447 }
1448 \cs_new:Npn \__file_parse_version:w #1 - #2 - #3 \s__file_stop {#1#2#3}

```

If the versions differ, then we try to give the user some guidance. This function starts by taking the engine name `\c_sys_engine_str` and replacing `tex` by `latex`, then building a command of the form: `kpsewhich -all -engine=<engine> <format>[-dev].fmt` to query the format files available. A shell is opened and each line is read into a sequence.

```

1449 \cs_new_protected:Npn \__file_mismatched_dependency_error:nn #1 #2
1450 {
1451   \exp_args:NNe \ior_shell_open:Nn \g__file_internal_ior
1452   {
1453     kpsewhich ~ --all ~
1454     --engine = \c_sys_engine_exec_str
1455     \c_space_tl \c_sys_engine_format_str
1456     \bool_lazy_and:nnT
1457     { \tl_if_exist_p:N \development@branch@name }
1458     { ! \tl_if_empty_p:N \development@branch@name }
1459     { -dev } .fmt

```

```

1460     }
1461     \seq_clear:N \l__file_tmp_seq
1462     \ior_map_inline:Nn \g__file_internal_ior
1463       { \seq_put_right:Nn \l__file_tmp_seq {##1} }
1464     \ior_close:N \g__file_internal_ior
1465     \msg_error:nnnn { kernel } { mismatched-support-file }
1466       {#1} {#2}

```

And finish by ending the current file.

```

1467     \tex_endinput:D
1468   }

```

Now define the actual error message:

```

1469 \msg_new:nnnn { kernel } { mismatched-support-file }
1470 {
1471   Mismatched-LaTeX-support-files-detected. \\
1472   Loading~'#2'~aborted!

```

`\c__kernel_expl_date_tl` may not exist, due to an older format, so only print the dates when the sentinel token list exists:

```

1473   \tl_if_exist:NT \c__kernel_expl_date_tl
1474   {
1475     \\ \\
1476     The~L3~programming~layer~in~the~LaTeX~format \\
1477     is~dated~\c__kernel_expl_date_tl,~but~in~your~TeX~
1478     tree~the~files~require \\ at~least~#1.
1479   }
1480 }
1481 {

```

The sequence containing the format files should have exactly one item: the format file currently being run. If that's the case, the cause of the error is not that, so print a generic help with some possible causes. If more than one format file was found, then print the list to the user, with appropriate indications of what's in the system and what's in the user tree.

```

1482   \int_compare:nNnTF { \seq_count:N \l__file_tmp_seq } > 1
1483   {
1484     The~cause~seems~to~be~an~old~format~file~in~the~user~tree. \\
1485     LaTeX~found~these~files:
1486     \seq_map_tokens:Nn \l__file_tmp_seq { \\~~~\use:n } \\
1487     Try~deleting~the~file~in~the~user~tree~then~run~LaTeX~again.
1488   }
1489   {

```

```

1490         The~most~likely~causes~are:
1491         \\~~~A~recent~format~generation~failed;
1492         \\~~~A~stray~format~file~in~the~user~tree~which~needs~
1493             to~be~removed~or~rebuilt;
1494         \\~~~You~are~running~a~manually~installed~version~of~#2 \\
1495         \\ \\ which~is~incompatible~with~the~version~in~LaTeX. \\
1496     }
1497     \\
1498     LaTeX~will~abort~loading~the~incompatible~support~files~
1499     but~this~may~lead~to \\ later~errors.~Please~ensure~that~
1500     your~LaTeX~format~is~correctly~regenerated.
1501 }

```

(`\_kernel_dependency_version_check:Nn` 以及其它的定义结束。)

### 3.7 Messages

```

1502 \msg_new:nnnn { kernel } { file-not-found }
1503 { File~'#1'~not~found. }
1504 {
1505     The~requested~file~could~not~be~found~in~the~current~directory,~
1506     in~the~TeX~search~path~or~in~the~LaTeX~search~path.
1507 }
1508 \msg_new:nnn { kernel } { file-list }
1509 {
1510     >~File~List~<
1511     #1 \\
1512     .....
1513 }
1514 \msg_new:nnnn { kernel } { filename-chars-lost }
1515 { #1~invalid~in~file~name.~Lost:~#2. }
1516 {
1517     There~was~an~invalid~token~in~the~file~name~that~caused~
1518     the~characters~following~it~to~be~lost.
1519 }
1520 \msg_new:nnnn { kernel } { filename-missing-endcsname }
1521 { Missing~\iow_char:N\\endcsname~inserted~in~filename. }
1522 {
1523     The~file~name~had~more~\iow_char:N\\csname~commands~than~
1524     \iow_char:N\\endcsname~ones.~LaTeX~will~add~the~missing~
1525     \iow_char:N\\endcsname~and~try~to~continue~as~best~as~it~can.
1526 }
1527 \msg_new:nnnn { kernel } { unbalanced-quote-in-filename }

```

```

1528 { Unbalanced-quotes-in-file-name-~'#1'. }
1529 {
1530   File-names-must-contain-balanced-numbers-of-quotes~(").
1531 }
1532 \msg_new:nnnn { kernel } { iow-indent }
1533 { Only-~#1 allows-~#2 }
1534 {
1535   The-command-~#2 can-only-be-used-in-messages~
1536   which-will-be-wrapped-using-~#1.
1537   \tl_if_empty:nF {#3} { ~ It-was-called-with-argument-~'#3'. }
1538 }

```

### 3.8 Functions delayed from earlier modules

<@@=sys>

`\c_sys_platform_str` Detecting the platform on LuaTeX is easy: for other engines, we use the fact that the two common cases have special null files. It is possible to probe further (see package `platform`), but that requires shell escape and seems unlikely to be useful. This is set up here as it requires file searching.

```

1539 \sys_if_engine luatex:TF
1540 {
1541   \str_const:Ne \c_sys_platform_str
1542   { \tex_directlua:D { tex.print(os.type) } }
1543 }
1544 {
1545   \file_if_exist:nTF { nul: }
1546   {
1547     \file_if_exist:nF { /dev/null }
1548     { \str_const:Nn \c_sys_platform_str { windows } }
1549   }
1550   {
1551     \file_if_exist:nT { /dev/null }
1552     { \str_const:Nn \c_sys_platform_str { unix } }
1553   }
1554 }
1555 \cs_if_exist:NF \c_sys_platform_str
1556 { \str_const:Nn \c_sys_platform_str { unknown } }

```

(`\c_sys_platform_str` 定义结束。这个变量被记录在第??页。)

`\sys_if_platform_unix_p:` We can now set up the tests.

`\sys_if_platform_unix:TF` 1557 `\clist_map_inline:nn { unix , windows }`

`\sys_if_platform_windows_p:`

`\sys_if_platform_windows:TF`

```

1558 {
1559     \__file_const:nn { sys_if_platform_ #1 }
1560     { \str_if_eq_p:Vn \c_sys_platform_str { #1 } }
1561 }

```

(*\sys\_if\_platform\_unix:TF* 和 *\sys\_if\_platform\_windows:TF* 定义结束。这些函数被记录在第??页。)

```

1562 \</package>

```

## 索引

斜体数字指向相应条目描述的页面，下划线数字指向定义的代码行，其它的都指向使用条目的页面。

Symbols			
\#	480	\char_set_catcode_space:n	1401
\%	482	clist 命令:	
\\	90, 485, 1471, 1475, 1476, 1478, 1484, 1486, 1491, 1492, 1494, 1495, 1497, 1499, 1511, 1521, 1523, 1524, 1525	\clist_if_exist:NTF	1248, 1369
\{	479	\clist_map_inline:nn	1557
\}	481	\contextversion	11, 41, 264, 284
\_	486, 1495	cs 命令:	
\^	394, 433	\cs:w	52, 788, 927
\~	483	\cs_end:	53, 788, 803, 806, 807, 916, 927
A		\cs_generate_variant:Nn	22, 27, 61, 86, 104, 106, 108, 279, 307, 315, 331, 343, 345, 347, 374, 377, 378, 391, 397, 398, 401, 404, 508, 865, 908, 1016, 1030, 1033, 1046, 1056, 1100, 1118, 1121, 1124, 1127, 1157, 1225, 1232, 1245, 1258, 1288, 1310, 1316, 1363
\AtBeginDocument	70, 1383	\cs_gset_eq:NN	101, 340
B		\cs_gset_protected:Npn	44, 215, 287
bool 命令:		\cs_if_exist:NTF	11, 15, 41, 264, 268, 284, 756, 937, 1381, 1434, 1555
\bool_if:NTF	700, 707	\cs_if_exist_use:N	477
\bool_lazy_and:nnTF	252, 1456	\cs_new:Npe	450, 462, 988, 1289
\bool_lazy_any:nTF	1059	\cs_new:Npn	199, 201, 276, 405, 413, 451, 468, 523, 574, 583, 602, 603, 611, 617, 625, 635, 640, 646, 652, 725, 727, 729, 777, 785, 791, 798, 799, 805, 807, 814, 819, 827, 837, 839, 848, 850, 851, 853, 903, 909,
\bool_lazy_or:nnTF	1004		
\bool_new:N	428		
\bool_set_false:N	534, 676, 684, 692, 702, 709		
\bool_set_true:N	662		
C			
char 命令:			
\l_char_active_seq	3		

914, 922, 931, 946, 952, 956, 962, 964, 975, 976, 977, 979, 997, 999, 1028, 1031, 1034, 1039, 1044, 1047, 1049, 1057, 1071, 1081, 1091, 1098, 1103, 1110, 1180, 1286, 1305, 1311, 1317, 1322, 1328, 1342, 1380, 1448	587, 589, 629, 695, 726, 747, 759, 787, 795, 886, 887, 888, 926, 1277, 1402
\cs_new_eq:NN . . . . 21, 43, 133, 278, 286, 406, 902, 1169, 1212, 1273, 1279	\exp_args:Nc . . . . . 210
\cs_new_protected:Npe . . 62, 878, 1246	\exp_args:Ne 115, 118, 354, 357, 423, 779, 781, 905, 919, 948, 1036, 1045, 1066, 1076, 1086, 1099, 1287, 1313
\cs_new_protected:Npn . . . . . 21, 25, 39, 50, 71, 77, 93, 105, 107, 109, 121, 122, 123, 150, 152, 163, 165, 184, 186, 188, 203, 205, 207, 213, 220, 229, 231, 233, 238, 278, 282, 294, 308, 316, 322, 332, 344, 346, 348, 360, 361, 362, 372, 375, 379, 385, 392, 399, 402, 414, 445, 456, 474, 509, 532, 544, 563, 567, 656, 672, 681, 689, 698, 705, 711, 860, 896, 1011, 1116, 1119, 1122, 1125, 1144, 1152, 1220, 1226, 1233, 1234, 1239, 1259, 1274, 1280, 1352, 1357, 1364, 1365, 1366, 1393, 1406, 1418, 1428, 1430, 1432, 1441, 1449	\exp_args:Nee . . . . . 1173, 1344
\cs_set:Npe . . . 479, 480, 481, 482, 483	\exp_args:Neee . . . . . 1051
\cs_set:Npn . . . . . 421, 743	\exp_args:Nf . . . . . 621, 679
\cs_set_eq:NN . . . . . . . . . 485, 486, 487, 488, 490, 492, 493	\exp_args:NNc . . . . . 40, 193, 194, 283
\cs_set_protected:Npn . . . . . . . . . . 400, 403, 542, 623, 723, 739	\exp_args:NNe . . . . . 1371, 1385, 1451
\cs_to_str:N . . . . . 406	\exp_args:Nne . . . . . 912, 963, 998
\cs_undefine:N . . . . . 46, 289	\exp_args:NNf . . . . . 39, 282, 505
\csname . . . . . 53	\exp_args:Nnne . . . . . 1293
<b>E</b>	\exp_args:NNNv . . . . . 196
else 命令:	\exp_args:Nno . . . . . 167, 895
\else: . . . . . 15, 140, 143, 146, 918, 925, 1186, 1195, 1206	\exp_args:No . . 226, 417, 513, 883, 983
\endcsname . . . . . 53	\exp_args:Nv . 655, 731, 870, 1431, 1436
exp 命令:	\exp_args:NVV . . . . . 566
\exp:w . . . . . 44, 588	\exp_args_generate:n . . . . . 1304
\exp_after:wN . . . . . 40, 224, 242, 283, 518, 521, 571, 580, 583, 586,	\exp_end_continue_f:w . . . . . 588
	\exp_last_unbraced:NNNNo . . . . . 797
	\exp_last_unbraced:NNo . . . . . 420
	\exp_not:N . . . . . 13, 64, 65, 69, 880, 883, 884, 886, 887, 888, 889, 892, 893, 990, 992, 1248, 1249, 1250, 1251, 1252, 1255, 1256, 1291, 1293, 1295, 1298, 1301
	\exp_not:n . . . . . 13, 376, 395
	\exp_stop_f: . . . . . . . . . 44, 40, 283, 576, 590, 602, 812
	\ExplFileDate . . . . 1396, 1411, 1425, 1429
	\ExplFileDescription . . . . . 1395, 1408
	\ExplFileExtension . . . . 1398, 1413, 1422
	\ExplFileName . . . . . 1397, 1412, 1421
	\ExplFileVersion . . . . . 1399, 1414, 1423
	<b>F</b>
	fi 命令:
	\fi: . . . . . 15, 55, 142, 145, 148, 225, 243, 528, 569, 578, 599, 609, 613, 620, 628, 823, 827, 830, 880, 893, 920, 929, 1188, 1197, 1208

\file_name .....	16	\__file_hex_dump:n .....	
file 内部命令:		.....	<a href="#">1049</a> , <a href="#">1099</a> , <a href="#">1103</a> , <a href="#">1110</a>
\l__file_base_name_tl .....	<a href="#">763</a>	\__file_hex_dump_auxi:nnn .....	
\__file_compare_timestamp:nnN ..		.....	<a href="#">1049</a> , <a href="#">1051</a> , <a href="#">1057</a>
.....	<a href="#">1170</a> , <a href="#">1173</a> , <a href="#">1180</a>	\__file_hex_dump_auxii:nnnn .....	
\__file_const:nn .....	<a href="#">1559</a>	.....	<a href="#">1049</a> , <a href="#">1066</a> , <a href="#">1071</a>
\__file_details:nn .....		\__file_hex_dump_auxiii:nnnn .....	
.....	<a href="#">1028</a> , <a href="#">1029</a> , <a href="#">1032</a> , <a href="#">1034</a>	.....	<a href="#">1049</a> , <a href="#">1074</a> , <a href="#">1076</a> , <a href="#">1081</a>
\__file_details_aux:nn .....		\__file_hex_dump_auxiiv:nnn ...	<a href="#">1049</a>
.....	<a href="#">1028</a> , <a href="#">1036</a> , <a href="#">1039</a> , <a href="#">1067</a>	\__file_hex_dump_auxiv:nnn .....	
\l__file_dir_str ....	<a href="#">765</a> , <a href="#">1268</a> , <a href="#">1269</a>	.....	<a href="#">1084</a> , <a href="#">1086</a> , <a href="#">1091</a>
\__file_ext_check:nn ...	<a href="#">944</a> , <a href="#">970</a> , <a href="#">977</a>	\__file_id_info_auxi:w .....	
\__file_ext_check:nnn .....	<a href="#">992</a> , <a href="#">997</a>	.....	<a href="#">1393</a> , <a href="#">1404</a> , <a href="#">1406</a>
\__file_ext_check:nnnn .....	<a href="#">998</a> , <a href="#">999</a>	\__file_id_info_auxii:w .....	
\__file_ext_check:nnnw ....	<a href="#">983</a> , <a href="#">988</a>	.....	<a href="#">72</a> , <a href="#">1393</a> , <a href="#">1416</a> , <a href="#">1418</a>
\__file_ext_check:nnw ..	<a href="#">978</a> , <a href="#">979</a> , <a href="#">986</a>	\__file_id_info_auxiii:w .....	
\l__file_ext_str ....	<a href="#">765</a> , <a href="#">1268</a> , <a href="#">1271</a>	.....	<a href="#">1393</a> , <a href="#">1426</a> , <a href="#">1428</a>
\__file_full_name:n ....	<a href="#">903</a> , <a href="#">905</a> , <a href="#">909</a>	\__file_if_recursion_tail_-	
\__file_full_name_assign:nnnNNN		break:NN .....	<a href="#">775</a>
.....	<a href="#">1355</a> , <a href="#">1357</a>	\__file_if_recursion_tail_stop:N	<a href="#">775</a>
\__file_full_name_aux:n .....		\__file_if_recursion_tail_stop_-	
.....	<a href="#">903</a> , <a href="#">912</a> , <a href="#">914</a> , <a href="#">963</a> , <a href="#">998</a>	do:Nn .....	<a href="#">775</a>
\__file_full_name_aux:nN	<a href="#">903</a> , <a href="#">948</a> , <a href="#">962</a>	\__file_if_recursion_tail_stop_-	
\__file_full_name_aux:Nnn .....		do:nn .....	<a href="#">776</a>
.....	<a href="#">903</a> , <a href="#">936</a> , <a href="#">940</a> , <a href="#">946</a>	\__file_input:n .....	
\__file_full_name_aux:nnN .....		..	<a href="#">1223</a> , <a href="#">1229</a> , <a href="#">1239</a> , <a href="#">1242</a> , <a href="#">1246</a> , <a href="#">1258</a>
.....	<a href="#">903</a> , <a href="#">963</a> , <a href="#">964</a>	\__file_input_pop: .....	
\__file_full_name_auxi:nn .....		.....	<a href="#">1239</a> , <a href="#">1256</a> , <a href="#">1274</a> , <a href="#">1279</a>
.....	<a href="#">903</a> , <a href="#">919</a> , <a href="#">922</a>	\__file_input_pop:nnn	<a href="#">1239</a> , <a href="#">1277</a> , <a href="#">1280</a>
\__file_full_name_auxii:nn .....		\__file_input_push:n .....	
.....	<a href="#">903</a> , <a href="#">912</a> , <a href="#">931</a>	.....	<a href="#">1239</a> , <a href="#">1251</a> , <a href="#">1259</a> , <a href="#">1273</a>
\__file_full_name_slash:n .....		\__file_input_raw:nn	<a href="#">1286</a> , <a href="#">1287</a> , <a href="#">1289</a>
.....	<a href="#">903</a> , <a href="#">949</a> , <a href="#">952</a>	\g__file_internal_ior .....	
\__file_full_name_slash:nw	<a href="#">954</a> , <a href="#">956</a>	.....	<a href="#">1027</a> , <a href="#">1451</a> , <a href="#">1462</a> , <a href="#">1464</a>
\__file_full_name_slash:w .....	<a href="#">903</a>	\l__file_internal_tl .	<a href="#">733</a> , <a href="#">1276</a> , <a href="#">1277</a>
\l__file_full_name_tl ..	<a href="#">763</a> , <a href="#">868</a> ,	\__file_kernel_dependency_-	
<a href="#">871</a> , <a href="#">1222</a> , <a href="#">1223</a> , <a href="#">1228</a> , <a href="#">1229</a> , <a href="#">1241</a> , <a href="#">1242</a>		compare:nnn	<a href="#">1430</a> , <a href="#">1436</a> , <a href="#">1439</a> , <a href="#">1441</a>
\__file_get_aux:nnN ....	<a href="#">860</a> , <a href="#">870</a> , <a href="#">878</a>	\__file_list:N .	<a href="#">1364</a> , <a href="#">1364</a> , <a href="#">1365</a> , <a href="#">1366</a>
\__file_get_details:nnN .....		\__file_list_aux:n .	<a href="#">1364</a> , <a href="#">1377</a> , <a href="#">1380</a>
..	<a href="#">1116</a> , <a href="#">1129</a> , <a href="#">1133</a> , <a href="#">1137</a> , <a href="#">1141</a> , <a href="#">1144</a>	\c__file_marker_tl .	<a href="#">55</a> , <a href="#">859</a> , <a href="#">884</a> , <a href="#">897</a>
\__file_get_do:Nw .....	<a href="#">860</a> , <a href="#">886</a> , <a href="#">896</a>	\__file_md5five_hash:n	<a href="#">1028</a> , <a href="#">1045</a> , <a href="#">1047</a>
\__file_get_full_name_search:nN	<a href="#">1011</a>		

\__file_mismatched_dependency_-	\__file_parse_full_name_auxi:nN
error:nn ..... 1446, 1449, 1449	..... 1313, 1317, 1317
\__file_name_cleanup:w . 903, 971, 975	\__file_parse_full_name_base:nw
\__file_name_end: . 903, 942, 975, 976	..... 69, 1325, 1328, 1328, 1340
\__file_name_expand:n .. 777, 782, 785	\__file_parse_full_name_tidy:nnnN
\__file_name_expand_cleanup:Nw .	.... 69, 1335, 1336, 1338, 1342, 1342
..... 53, 777, 787, 791	\__file_parse_version:w .....
\__file_name_expand_cleanup:w ..	..... 1430, 1444, 1445, 1448
..... 52, 777, 795, 798	\__file_quark_if_nil:n ..... 772
\__file_name_expand_end: .....	\__file_quark_if_nil:nTF .....
..... 52, 53, 777,	..... 772, 841, 855, 981, 990
789, 791, 794, 799, 803, 805, 806, 808	\__file_quark_if_nil_p:n ..... 772
\__file_name_expand_error:Nw ...	\g__file_record_seq ..... 66,
..... 52, 53, 777, 794, 805	70, 71, 762, 1250, 1374, 1388, 1389
\__file_name_expand_error_aux:Nw	\__file_size:n ..... 902, 902, 919
..... 53, 777, 806, 807	\g__file_stack_seq 66, 737, 1261, 1276
\__file_name_ext_check:nn ..... 903	\__file_str_cmp:nn . 1169, 1169, 1201
\__file_name_ext_check:nnn .... 903	\__file_timestamp:n .....
\__file_name_ext_check:nnnn .... 903	..... 1170, 1202, 1203, 1212
\__file_name_ext_check:nnnw .... 903	\__file_tmp:w . 739, 743, 747, 753, 759
\__file_name_ext_check:nnw .... 903	\l__file_tmp_seq .....
\__file_name_quote:nw .. 851, 852, 853	769, 1368, 1371, 1374, 1375, 1377,
\l__file_name_str ... 765, 1268, 1270	1385, 1390, 1461, 1463, 1482, 1486
\__file_name_strip_quotes:n ....	file 命令:
..... 777, 781, 814	\file_compare_timestamp:nNn ....
\__file_name_strip_quotes:nnn .. 777	..... 1170, 1178
\__file_name_strip_quotes:nnnw . 777	\file_compare_timestamp:nNnTF ..
\__file_name_strip_quotes:nw ...	..... 18, 1170
..... 816, 819, 825, 828	\file_compare_timestamp_p:nNn ..
\__file_name_strip_quotes_-	..... 18, 1170
end:wNwn ..... 822, 827	\g_file_curr_dir_str .....
\__file_name_trim_spaces:n ....	..... 15, 734, 1263, 1269, 1282
..... 777, 779, 837	\g_file_curr_ext_str .....
\__file_name_trim_spaces:nw ....	..... 15, 734, 1265, 1271, 1284
..... 777, 838, 839	\g_file_curr_name_str .....
\__file_name_trim_spaces_aux:n .	..... 15, 734, 1264, 1270, 1283
..... 777, 844, 848	\file_full_name:n ..... 18,
\__file_name_trim_spaces_aux:w .	903, 903, 908, 1020, 1037, 1045,
..... 777, 849, 850	1052, 1099, 1174, 1175, 1215, 1287
\__file_parse_full_name_area:nw	\file_get:nnN 20, 860, 860, 865, 866, 877
..... 68, 1317, 1319, 1322, 1326	\file_get:nnNTF ..... 20, 860, 862





\int_gincr:N . . . . .	209	ior 命令:	
\int_if_odd:nTF . . . . .	831	\ior_close:N	4, 5, 52, <u>93</u> , 93, 104, 1464
\int_new:N . . . . .	407, 410, 412, 425	\ior_get:NN . . .	5–9, <u>150</u> , 150, 154, 230
\int_set:Nn . . . . .	169, 171, 387,	\ior_get:NNTF . . . . .	6, <u>150</u> , 151
389, 408, 418, 431, 478, 484, 496, 501		\ior_get_term:nN . . . . .	9, <u>184</u> , 184
\int_set_eq:NN . . . . .	882	\ior_if_eof:N . . . . .	29, 134
\int_step_inline:nnn . . . . .	8, 261	\ior_if_eof:NTF . . . . .	
\int_sub:Nn . . . . .	693	. . . . .	9, <u>134</u> , 156, 176, 216, 235
\int_use:N . . . . .	172, 211, 383	\ior_if_eof_p:N . . . . .	9, <u>134</u>
\int_value:w . . . . .	572, 581	\ior_log:N . . . . .	5, <u>105</u> , 107, 108
\int_zero:N . . . . .	538	\ior_log_list: . . . . .	5, <u>121</u> , 122
\c_zero_int . . . . .	882, 1204	\ior_map_break: . . . . .	8,
ior 内部命令:		<u>199</u> , 199, 200, 202, 217, 224, 236, 242	
\l__ior_file_name_tl . . . . .	28, 31, 33	\ior_map_break:n . . . . .	9, <u>199</u> , 201
\__ior_get:NN . . . . .	<u>150</u> , 152, 159, 185, 204	\ior_map_inline:Nn	7, 8, <u>203</u> , 203, 1462
\__ior_get_term:NnN	<u>184</u> , 185, 187, 188	\ior_map_variable:NNn . . .	8, <u>229</u> , 229
\l__ior_internal_tl . . . . .		\ior_new:N . . .	3, <u>21</u> , 21, 22, 23, 24, 1027
. . . . .	3, 113, 116, 222, 226	\ior_open:Nn . .	4, <u>66</u> , <u>25</u> , 25, 27, 29, 38
\__ior_list:N . . . . .	<u>121</u> , 121, 122, 123	\ior_open:NnTF . . . . .	4, 26, <u>29</u>
\__ior_map_inline:NNn . . . . .		\ior_shell_open:Nn . . .	4, <u>71</u> , 71, 1451
. . . . .	<u>203</u> , 204, 206, 207	\ior_show:N . . . . .	5, <u>105</u> , 105, 106
\__ior_map_inline:NNNn . . . . .	<u>203</u> , 210, 213	\ior_show_list: . . . . .	5, <u>121</u> , 121
\__ior_map_inline_loop:NNN . . . . .	<u>203</u> , 216, 220, 227	\ior_str_get:NN . . . . .	
\__ior_map_variable:NNNn . . . . .	<u>229</u> , 230, 232, 233	. . . . .	5, 7, 9, <u>163</u> , 163, 174, 232
\__ior_map_variable_loop:NNNn . . . . .	<u>229</u> , 235, 238, 245	\ior_str_get:NNTF . . . . .	7, <u>163</u> , 164
\__ior_new:N . . . . .	<u>23</u> , <u>39</u> , 39, 43, 44, 56	\ior_str_get_term:nN . . . .	9, <u>184</u> , 186
\__ior_new_aux:N . . . . .	43, 47	\ior_str_map_inline:Nn	7, 8, <u>203</u> , 205
\__ior_open_stream:Nn . . .	<u>50</u> , 54, 58, 62	\ior_str_map_variable:NNn	8, <u>229</u> , 231
\__ior_shell_open:nN . . .	<u>71</u> , 74, 77, 86	\g_tmpa_ior . . . . .	14, <u>23</u>
\__ior_show:NN . . . . .	<u>105</u> , 105, 107, 109	\g_tmpb_ior . . . . .	14, <u>23</u>
\__ior_str_get:NN . . . . .	<u>163</u> , 165, 179, 187, 206	iorw 内部命令:	
\l__ior_stream_tl . . . . .	6, 53, 57, 64	\l__iow_file_name_tl	<u>293</u> , 296, 300, 304
\g__ior_streams_prop . . . . .	26, 7, 65, 98, 113, 128	\__iow_indent:n . . . . .	40, <u>456</u> , 462, 488
\g__ior_streams_seq . . . .	5, 53, 99, 100	\__iow_indent_error:n	40, <u>456</u> , 468, 493
\c__ior_term_ior	4, 21, 95, 101, 137, 194	\l__iow_indent_int . . . . .	
\c__ior_term_noprompt_ior .	<u>183</u> , 193	. . . . .	<u>424</u> , 538, 556, 668, 685, 693
		\l__iow_indent_tl . . . . .	
		. . . . .	<u>424</u> , 539, 555, 667, 686, 694, 695
		\l__iow_internal_tl . . . .	<u>248</u> , 352, 355
		\l__iow_line_break_bool	<u>428</u> , 534,
			662, 676, 684, 692, 700, 702, 707, 709

\l__iow_line_part_tl .....	\__iow_wrap_break_loop:w .....
... <a href="#">43</a> , <a href="#">45</a> , <a href="#">47</a> , <a href="#">426</a> , 536, 548, 569,	..... <a href="#">623</a> , 638, 646, 650
627, 630, 661, 675, 677, 683, 691, 714	\__iow_wrap_break_none:w <a href="#">623</a> , 637, 640
\l__iow_line_target_int .....	\__iow_wrap_chunk:nw .....
. <a href="#">48</a> , <a href="#">410</a> , 496, 498, 501, 663, 668, 703	540, <a href="#">542</a> , 544, 678, 679, 687, 696, 703
\l__iow_line_tl .....	\__iow_wrap_do: .....
..... <a href="#">426</a> , 535, 552, 642, 658,	504, <a href="#">509</a> , 509
674, 675, 683, 691, 713, 714, 719, 721	\__iow_wrap_end:n .....
\__iow_list:N .....	<a href="#">698</a> , 705
360, 360, 361, 362	\__iow_wrap_end_chunk:w .....
\__iow_new:N ..	..... <a href="#">43</a> , 560, <a href="#">567</a> , 617, 659
<a href="#">282</a> , 282, 286, 287, 302	\c__iow_wrap_end_marker_tl <a href="#">430</a> , 514
\__iow_new_aux:N .....	\__iow_wrap_fix_newline:w .....
286, 290	..... <a href="#">509</a> , 518, 523, 530
\l__iow_newline_tl .....	\__iow_wrap_indent:n .....
..... <a href="#">409</a> , 494, 495, 497, 500, 718	<a href="#">681</a> , 681
\l__iow_one_indent_int . <a href="#">411</a> , 685, 693	\c__iow_wrap_indent_marker_tl ..
\l__iow_one_indent_tl ... <a href="#">38</a> , <a href="#">411</a> , 686	..... <a href="#">430</a> , 464
\__iow_open_stream:Nn .....	\__iow_wrap_line:nw .....
..... <a href="#">294</a> , 300, 304, 308, 315	..... <a href="#">43</a> , <a href="#">46</a> , 554, 558, <a href="#">567</a> , 567, 666
\__iow_set_indent:n . <a href="#">38</a> , <a href="#">411</a> , 414, 423	\__iow_wrap_line_aux:Nw <a href="#">567</a> , 577, 583
\__iow_shell_open:nN <a href="#">316</a> , 319, 322, 331	\__iow_wrap_line_end:NnnnnnnN .
\__iow_show:NN .... <a href="#">344</a> , 344, 346, 348	..... <a href="#">567</a> , 586, 603
\l__iow_stream_tl . <a href="#">259</a> , 299, 303, 310	\__iow_wrap_line_end:nw .....
\g__iow_streams_prop .....	..... <a href="#">45</a> , <a href="#">567</a> , 608, 611, 643, 644, 653
..... <a href="#">34</a> , <a href="#">260</a> , 311, 337, 352, 367	\__iow_wrap_line_loop:w .....
\g__iow_streams_seq <a href="#">258</a> , 299, 338, 339	..... <a href="#">567</a> , 571, 574, 580
\__iow_tmp:w .....	\__iow_wrap_line_seven:nnnnnnn .
..... <a href="#">45</a> , 542, 566, 623, 655, 723, 731	..... <a href="#">567</a> , 598, 602
\__iow_unindent:w <a href="#">38</a> , <a href="#">411</a> , 413, 421, 695	\c__iow_wrap_marker_tl <a href="#">39</a> , <a href="#">43</a> , <a href="#">430</a> , 566
\__iow_use_i_delimit_by_s-	\__iow_wrap_newline:n .....
stop:nw .....	<a href="#">698</a> , 698
<a href="#">276</a> , 276, 527	\c__iow_wrap_newline_marker_tl .
\__iow_with:nNnn ..	..... <a href="#">42</a> , <a href="#">430</a> , 529
<a href="#">379</a> , 383, 385, 391	\__iow_wrap_next:nw .....
\__iow_wrap_allow_break: .....	..... <a href="#">542</a> , 549, 563, 621, 663
..... <a href="#">39</a> , <a href="#">445</a> , 450, 487	\__iow_wrap_next_line:w <a href="#">615</a> , <a href="#">656</a> , 656
\__iow_wrap_allow_break:n . <a href="#">672</a> , 672	\__iow_wrap_start:w .... <a href="#">509</a> , 521, 532
\__iow_wrap_allow_break_error: .	\__iow_wrap_store_do:n .....
..... <a href="#">39</a> , <a href="#">445</a> , 451, 492	..... 614, 701, 708, <a href="#">711</a> , 711
\c__iow_wrap_allow_break_marker-	\l__iow_wrap_tl .....
tl .....	..... <a href="#">41</a> , <a href="#">42</a> , <a href="#">48</a> , <a href="#">429</a> , 491, 506,
<a href="#">430</a> , 450	511, 513, 516, 518, 521, 537, 715, 717
\__iow_wrap_break:w .... 609, <a href="#">623</a> , 625	\__iow_wrap_trim:N .....
\__iow_wrap_break_end:w .....	..... <a href="#">49</a> , 644, 675, 701, 708, <a href="#">723</a> , 725
..... <a href="#">45</a> , <a href="#">623</a> , 632, 652	\__iow_wrap_trim:w .... <a href="#">723</a> , 726, 727
\__iow_wrap_break_first:w .....	
..... <a href="#">623</a> , 629, 635	

\\_\_iow\_wrap\_trim\_aux:w . [723](#), [728](#), [729](#)  
 \\_\_iow\_wrap\_unindent:n .... [681](#), [689](#)  
 \c\_\_iow\_wrap\_unindent\_marker\_tl  
 ..... [430](#), [466](#)

iow 命令:

\iow\_char:N .....  
 . [11](#), [406](#), [406](#), [1521](#), [1523](#), [1524](#), [1525](#)  
 \iow\_close:N .. [4](#), [5](#), [298](#), [332](#), [332](#), [343](#)  
 \iow\_indent:n ..... [13](#),  
[14](#), [40](#), [41](#), [456](#), [456](#), [459](#), [471](#), [488](#), [493](#)  
 \l\_iow\_line\_count\_int .....  
 ..... [13](#), [14](#), [41](#), [407](#), [497](#), [502](#), [540](#)  
 \iow\_log:N ..... [5](#), [344](#), [346](#), [347](#)  
 \iow\_log:n .... [10](#), [399](#), [399](#), [400](#), [401](#)  
 \iow\_log\_list: ..... [5](#), [360](#), [361](#)  
 \iow\_new:N .. [3](#), [278](#), [278](#), [279](#), [280](#), [281](#)  
 \iow\_newline: ..... [10](#),  
[11](#), [13](#), [36](#), [405](#), [405](#), [485](#), [494](#), [500](#), [1380](#)  
 \iow\_now:Nn ..... [10](#), [11](#),  
[392](#), [392](#), [397](#), [398](#), [399](#), [400](#), [402](#), [403](#)  
 \iow\_open:Nn ..... [4](#), [294](#), [294](#), [307](#)  
 \iow\_shell\_open:Nn ..... [5](#), [316](#), [316](#)  
 \iow\_shipout:Nn .....  
 ..... [10](#), [11](#), [36](#), [375](#), [375](#), [377](#), [378](#)  
 \iow\_shipout\_e:Nn [10](#), [11](#), [372](#), [372](#), [374](#)  
 \iow\_shipout\_x:Nn ..... [36](#)  
 \iow\_show:N ..... [5](#), [344](#), [344](#), [345](#)  
 \iow\_show\_list: ..... [5](#), [360](#), [360](#)  
 \iow\_term:n .. [9](#), [10](#), [399](#), [402](#), [403](#), [404](#)  
 \iow\_wrap:nnnN ..... [10](#), [13](#),  
[14](#), [41](#), [448](#), [454](#), [459](#), [471](#), [474](#), [474](#), [508](#)  
 \iow\_wrap\_allow\_break: .....  
 .. [13](#), [14](#), [445](#), [445](#), [448](#), [454](#), [487](#), [492](#)  
 \iow\_wrap\_allow\_break:n ..... [39](#)  
 \c\_log\_iow . [14](#), [31](#), [249](#), [334](#), [399](#), [400](#)  
 \c\_term\_iow .....  
 .. [14](#), [31](#), [249](#), [278](#), [334](#), [340](#), [402](#), [403](#)  
 \g\_tmpa\_iow ..... [14](#), [280](#)  
 \g\_tmpb\_iow ..... [14](#), [280](#)

## K

kernel 内部命令:

\\_\_kernel\_chk\_defined:NTF . [111](#), [350](#)

\\_\_kernel\_dependency\_version\_-  
 check:Nn ..... [1430](#), [1430](#)  
 \\_\_kernel\_dependency\_version\_-  
 check:nn ..... [1430](#), [1431](#), [1432](#)  
 \c\_\_kernel\_expl\_date\_tl .....  
 ..... [74](#), [1434](#), [1437](#), [1473](#), [1477](#)  
 \\_\_kernel\_file\_input\_pop: [1239](#), [1279](#)  
 \\_\_kernel\_file\_input\_push:n ....  
 ..... [1239](#), [1273](#)  
 \\_\_kernel\_file\_missing:n .....  
 ..... [26](#), [1234](#), [1234](#), [1243](#)  
 \\_\_kernel\_file\_name\_quote:n ....  
[24](#), [69](#), [313](#), [851](#), [851](#), [892](#), [1255](#), [1301](#)  
 \\_\_kernel\_file\_name\_sanitize:n .  
[68](#), [297](#), [777](#), [777](#), [906](#), [1237](#), [1295](#), [1314](#)  
 \\_\_kernel\_ior\_open:Nn .....  
 ..... [24](#), [33](#), [50](#), [50](#), [61](#), [84](#)  
 \\_\_kernel\_iow\_open:Nn ..... [329](#)  
 \\_\_kernel\_iow\_with:Nnn .....  
 ..... [36](#), [379](#), [379](#), [394](#)  
 \g\_\_kernel\_prg\_map\_int . [209](#), [211](#), [218](#)  
 \\_\_kernel\_quark\_new\_conditional:Nn  
 ..... [772](#)  
 \\_\_kernel\_quark\_new\_test:N [775](#), [776](#)  
 \\_\_kernel\_str\_to\_other\_fast:n ..  
 ..... [417](#), [513](#)  
 \\_\_kernel\_tl\_set:Nn ..... [57](#), [296](#),  
[303](#), [416](#), [491](#), [494](#), [495](#), [511](#), [516](#),  
[674](#), [694](#), [713](#), [715](#), [1019](#), [1146](#), [1161](#)

## L

\loccount ..... [15](#), [268](#)

## M

msg 命令:

\msg\_critical:nn ..... [21](#)  
 \msg\_error:nn ..... [75](#), [320](#)  
 \msg\_error:nnn ..... [81](#), [326](#), [1236](#)  
 \msg\_error:nnnn ..... [447](#), [1465](#)  
 \msg\_error:nnnnn ..... [458](#)  
 \msg\_expandable\_error:nn ..... [801](#)  
 \msg\_expandable\_error:nnn [833](#), [1293](#)  
 \msg\_expandable\_error:nnnn [453](#), [810](#)

`\msg_expandable_error:nnnnn` .... 470  
`\msg_log:nnnnnn` ..... 122, 361, 1365  
`\msg_new:nnn` ..... 1508  
`\msg_new:nnnn` .....  
     87, 1469, 1502, 1514, 1520, 1527, 1532  
`\msg_show:nnnnnn` .... 121, 360, 1364  
`\msg_show_item_unbraced:nn` ....  
     ..... 26, 129, 368

## O

or 命令:  
     `\or:` .. 592, 593, 594, 595, 596, 597, 598

## P

`\par` ..... 6  
 prg 命令:  
     `\prg_break_point:Nn` ..... 217, 236  
     `\prg_do_nothing:` .....  
         ..... 68, 591, 888, 1308, 1347, 1349  
     `\prg_generate_conditional_-`  
         variant:Nnn 38, 877, 1025, 1130,  
         1134, 1138, 1142, 1167, 1178, 1219  
     `\prg_map_break:Nn` ..... 200, 202  
     `\prg_new_conditional:Npnn` .....  
         ..... 134, 1170, 1213  
     `\prg_new_protected_conditional:Npnn`  
         ..... 29, 154, 174,  
         866, 1017, 1128, 1132, 1136, 1140, 1158  
     `\prg_replicate:nn` ..... 423  
     `\prg_return_false:` .....  
         .... 36, 141, 157, 177, 875, 1022,  
         1149, 1164, 1187, 1196, 1207, 1216  
     `\prg_return_true:` ..... 55, 34,  
         139, 144, 147, 160, 180, 873, 1023,  
         1150, 1165, 1185, 1194, 1205, 1217  
 prop 命令:  
     `\prop_get:NnNTF` ..... 113, 352  
     `\prop_gput:Nnn` ..... 19, 65, 272, 311  
     `\prop_gremove:Nn` ..... 98, 337  
     `\prop_map_function:NN` ..... 128, 367  
     `\prop_new:N` ..... 7, 260  
`\protect` ..... 490

## Q

quark 内部命令:  
     `\q__file_nil` .. 771, 838, 852, 978, 984  
     `\q__file_recursion_stop` 773, 817, 828  
     `\q__file_recursion_tail` 773, 817, 821  
     `\q__iow_nil` ..... 277, 519, 526  
 quark 命令:  
     `\q_nil` ..... 954, 956  
     `\q_no_value` 6, 7, 16–18, 20, 151, 164,  
         863, 1014, 1117, 1120, 1123, 1126, 1155  
     `\quark_if_nil:nTF` ..... 958  
     `\quark_new:N` ..... 277, 771, 773, 774  
     `\q_stop` ..... 954, 956

## S

scan 内部命令:  
     `\s__file_stop` ..... 68, 743,  
         748, 770, 838, 839, 843, 850, 852,  
         853, 978, 979, 984, 986, 988, 1320,  
         1322, 1325, 1326, 1328, 1340, 1416,  
         1419, 1426, 1428, 1444, 1445, 1448  
     `\s__iow_mark` .....  
         274, 633, 640, 652, 726, 727, 728, 729  
     `\s__iow_stop` ..... 274,  
         276, 519, 560, 618, 656, 669, 726, 729  
 scan 命令:  
     `\scan_new:N` ..... 274, 275, 770  
     `\scan_stop:` ..... 57, 64, 69,  
         191, 310, 313, 885, 892, 924, 1255, 1301  
 seq 命令:  
     `\seq_clear:N` ..... 1368, 1461  
     `\seq_concat:NNN` ..... 1374  
     `\seq_count:N` ..... 1482  
     `\seq_gconcat:NNN` ..... 1387  
     `\seq_gpop:NN` ..... 1276  
     `\seq_gpop:NNTF` ..... 53, 299  
     `\seq_gpush:Nn` ..... 100, 339, 1261  
     `\seq_gput_right:Nn` ... 745, 752, 1250  
     `\seq_if_in:NnTF` ..... 99, 338  
     `\seq_map_break:n` ..... 936  
     `\seq_map_function:NN` ..... 1377  
     `\seq_map_tokens:Nn` ..... 935, 1486  
     `\seq_new:N` .. 5, 258, 737, 762, 768, 769



