Abstract

This is the reference documentation for the \texttt{expl3} programming environment. The \texttt{expl3} modules set up an experimental naming scheme for LaTeX commands, which allow the LaTeX programmer to systematically name functions and variables, and specify the argument types of functions.

The \TeX{} and \texttt{\LaTeX} primitives are all given a new name according to these conventions. However, in the main direct use of the primitives is not required or encouraged: the \texttt{expl3} modules define an independent low-level \LaTeX{}3 programming language.

At present, the \texttt{expl3} modules are designed to be loaded on top of \LaTeX{}2\epsilon. In time, a \LaTeX{}3 format will be produced based on this code. This allows the code to be used in \LaTeX{}2\epsilon packages now while a stand-alone \LaTeX{}3 is developed.

While \texttt{expl3} is still experimental, the bundle is now regarded as broadly stable. The syntax conventions and functions provided are now ready for wider use. There may still be changes to some functions, but these will be minor when compared to the scope of \texttt{expl3}.

New modules will be added to the distributed version of \texttt{expl3} as they reach maturity.

\texttt{\LaTeX{}3\ Project}\textsuperscript{*}

March 1, 2015

\textsuperscript{*}E-mail: latex-team@latex-project.org
# Contents

## I Introduction to expl3 and this document  
1. **Naming functions and variables**  
   1.1 Terminological inexactitude  
2. **Documentation conventions**  
3. **Formal language conventions which apply generally**  
4. **\TeX concepts not supported by \LaTeX3**

## II The l3bootstrap package: Bootstrap code  
1. **Using the \LaTeX3 modules**  
   1.1 Internal functions and variables

## III The l3names package: Namespace for primitives  
1. **Setting up the \LaTeX3 programming language**

## IV The l3basics package: Basic definitions  
1. **No operation functions**  
2. **Grouping material**  
3. **Control sequences and functions**  
   3.1 Defining functions  
   3.2 Defining new functions using parameter text  
   3.3 Defining new functions using the signature  
   3.4 Copying control sequences  
   3.5 Deleting control sequences  
   3.6 Showing control sequences  
   3.7 Converting to and from control sequences
4. **Using or removing tokens and arguments**  
   4.1 Selecting tokens from delimited arguments
5. **Predicates and conditionals**  
   5.1 Tests on control sequences  
   5.2 Engine-specific conditionals  
   5.3 Primitive conditionals
6 Internal kernel functions 24

V The \texttt{l3expan} package: Argument expansion 27
1 Defining new variants 27
2 Methods for defining variants 28
3 Introducing the variants 28
4 Manipulating the first argument 30
5 Manipulating two arguments 31
6 Manipulating three arguments 32
7 Unbraced expansion 33
8 Preventing expansion 34
9 Internal functions and variables 35

VI The \texttt{l3prg} package: Control structures 36
1 Defining a set of conditional functions 36
2 The boolean data type 38
3 Boolean expressions 40
4 Logical loops 41
5 Producing multiple copies 42
6 Detecting \TeX{}’s mode 42
7 Primitive conditionals 43
8 Internal programming functions 43

VII The \texttt{l3quark} package: Quarks 45
1 Introduction to quarks and scan marks 45
  1.1 Quarks ................................................................. 45
2 Defining quarks 46
3 Quark tests 46
4 Recursion 47
5 An example of recursion with quarks 48
6 Internal quark functions 48
7 Scan marks 49

VIII The \texttt{l3token} package: Token manipulation 50
1 All possible tokens 50
2 Character tokens 51
3 Generic tokens 54
4 Converting tokens 55
5 Token conditionals 55
6 Peeking ahead at the next token 59
7 Decomposing a macro definition 62

IX The \texttt{l3int} package: Integers 63
1 Integer expressions 63
2 Creating and initialising integers 64
3 Setting and incrementing integers 65
4 Using integers 66
5 Integer expression conditionals 66
6 Integer expression loops 68
7 Integer step functions 70
8 Formatting integers 70
9 Converting from other formats to integers 72
10 Viewing integers 73
X The \texttt{l3skip} package: Dimensions and skips

1. Creating and initialising \texttt{dim} variables
2. Setting \texttt{dim} variables
3. Utilities for dimension calculations
4. Dimension expression conditionals
5. Dimension expression loops
6. Using \texttt{dim} expressions and variables
7. Viewing \texttt{dim} variables
8. Constant dimensions
9. Scratch dimensions
10. Creating and initialising \texttt{skip} variables
11. Setting \texttt{skip} variables
12. Skip expression conditionals
13. Using \texttt{skip} expressions and variables
14. Viewing \texttt{skip} variables
15. Constant skips
16. Scratch skips
17. Inserting skips into the output
18. Creating and initialising \texttt{muskip} variables
19. Setting \texttt{muskip} variables
8 Using a single item 128
9 Viewing comma lists 128
10 Constant and scratch comma lists 128

XV The \texttt{l3prop} package: Property lists 130
1 Creating and initialising property lists 130
2 Adding entries to property lists 131
3 Recovering values from property lists 131
4 Modifying property lists 132
5 Property list conditionals 132
6 Recovering values from property lists with branching 133
7 Mapping to property lists 133
8 Viewing property lists 134
9 Scratch property lists 135
10 Constants 135
11 Internal property list functions 135

XVI The \texttt{l3box} package: Boxes 136
1 Creating and initialising boxes 136
2 Using boxes 137
3 Measuring and setting box dimensions 137
4 Box conditionals 138
5 The last box inserted 139
6 Constant boxes 139
7 Scratch boxes 139
8 Viewing box contents 139
9 Horizontal mode boxes 140
10 Vertical mode boxes 141
11 Primitive box conditionals 143

XVII The \texttt{l3coffins} package: Coffin code layer 144
1 Creating and initialising coffins 144
2 Setting coffin content and poles 144
3 Joining and using coffins 146
4 Measuring coffins 146
5 Coffin diagnostics 147
5.1 Constants and variables .............................. 147

XVIII The \texttt{l3color} package: Color support 148
1 Color in boxes 148

XIX The \texttt{l3msg} package: Messages 149
1 Creating new messages 149
2 Contextual information for messages 150
3 Issuing messages 151
4 Redirecting messages 153
5 Low-level message functions 154
6 Kernel-specific functions 156
7 Expandable errors 157
8 Internal \texttt{l3msg} functions 158

XX The \texttt{l3keys} package: Key–value interfaces 160
1 Creating keys 161
9 Floating point expressions 189
  9.1 Input of floating point numbers ................................. 189
  9.2 Precedence of operators ........................................... 190
  9.3 Operations ............................................................ 191

10 Disclaimer and roadmap 196

XXIII The \texttt{l3candidates} package: Experimental additions to \texttt{l3kernel} 199

1 Important notice 199
2 Additions to \texttt{l3basics} 199
3 Additions to \texttt{l3box} 200
  3.1 Affine transformations ............................................. 200
  3.2 Viewing part of a box ............................................... 202
  3.3 Internal variables .................................................. 202
4 Additions to \texttt{l3clist} 203
5 Additions to \texttt{l3coffins} 203
6 Additions to \texttt{l3file} 204
7 Additions to \texttt{l3fp} 205
8 Additions to \texttt{l3int} 205
9 Additions to \texttt{l3keys} 206
10 Additions to \texttt{l3msg} 206
11 Additions to \texttt{l3prg} 206
12 Additions to \texttt{l3prop} 207
13 Additions to \texttt{l3seq} 207
14 Additions to \texttt{l3skip} 208
15 Additions to \texttt{l3tl} 209
16 Additions to \texttt{l3tokens} 213

XXIV The \texttt{l3drivers} package: Drivers 215
1  Box clipping 215
2  Box rotation and scaling 216
3  Color support 216

Index 217
Part I
Introduction to expl3 and this document

This document is intended to act as a comprehensive reference manual for the expl3 language. A general guide to the \TeX3 programming language is found in expl3.pdf.

1 Naming functions and variables

\TeX3 does not use \texttt{\@} as a “letter” for defining internal macros. Instead, the symbols \texttt{\_} and \texttt{:} are used in internal macro names to provide structure. The name of each \textit{function} is divided into logical units using \texttt{\_}, while \texttt{:} separates the \textit{name} of the function from the \textit{argument specifier} (“arg-spec”). This describes the arguments expected by the function. In most cases, each argument is represented by a single letter. The complete list of arg-spec letters for a function is referred to as the \textit{signature} of the function.

Each function name starts with the \textit{module} to which it belongs. Thus apart from a small number of very basic functions, all expl3 function names contain at least one underscore to divide the module name from the descriptive name of the function. For example, all functions concerned with comma lists are in module \texttt{clist} and begin \texttt{\clist\_}.

Every function must include an argument specifier. For functions which take no arguments, this will be blank and the function name will end \texttt{:}. Most functions take one or more arguments, and use the following argument specifiers:

\begin{itemize}
  \item \texttt{D} The D specifier means \textit{do not use}. All of the \TeX primitives are initially \texttt{\let} to a D name, and some are then given a second name. Only the kernel team should use anything with a D specifier!
  \item \texttt{N} and \texttt{n} These mean \textit{no manipulation}, of a single token for \texttt{N} and of a set of tokens given in braces for \texttt{n}. Both pass the argument through exactly as given. Usually, if you use a single token for an \texttt{n} argument, all will be well.
  \item \texttt{c} This means \textit{csname}, and indicates that the argument will be turned into a csname before being used. So \texttt{\foo:c \{ArgumentOne\}} will act in the same way as \texttt{\foo:n \{ArgumentOne\}}.
  \item \texttt{V} and \texttt{v} These mean \textit{value of variable}. The V and v specifiers are used to get the content of a variable without needing to worry about the underlying \TeX structure containing the data. A V argument will be a single token (similar to \texttt{N}), for example \texttt{\foo:V \MyVariable}; on the other hand, using v a csname is constructed first, and then the value is recovered, for example \texttt{\foo:v \{MyVariable\}}.
  \item \texttt{o} This means \textit{expansion once}. In general, the V and v specifiers are favoured over o for recovering stored information. However, o is useful for correctly processing information with delimited arguments.
\end{itemize}
x The x specifier stands for \textit{exhaustive expansion}: every token in the argument is fully expanded until only unexpandable ones remain. The \TeX \texttt{edef} primitive carries out this type of expansion. Functions which feature an x-type argument are in general \textit{not} expandable, unless specifically noted.

f The f specifier stands for \textit{full expansion}, and in contrast to x stops at the first non-expandable item (reading the argument from left to right) without trying to expand it. For example, when setting a token list variable (a macro used for storage), the sequence

\begin{verbatim}
\tl_set:Nn \l_mya_tl { A }
\tl_set:Nn \l_myb_tl { B }
\tl_set:Nf \l_mya_tl { \l_mya_tl \l_myb_tl }
\end{verbatim}

will leave \l_mya_tl with the content A\l_myb_tl, as A cannot be expanded and so terminates expansion before \l_myb_tl is considered.

T and F For logic tests, there are the branch specifiers T (true) and F (false). Both specifiers treat the input in the same way as n (no change), but make the logic much easier to see.

p The letter p indicates \TeX \texttt{parameters}. Normally this will be used for delimited functions as expl3 provides better methods for creating simple sequential arguments.

w Finally, there is the w specifier for \textit{weird} arguments. This covers everything else, but mainly applies to delimited values (where the argument must be terminated by some arbitrary string).

Notice that the argument specifier describes how the argument is processed prior to being passed to the underlying function. For example, \texttt{\foo:c} will take its argument, convert it to a control sequence and pass it to \texttt{\foo:N}.

Variables are named in a similar manner to functions, but begin with a single letter to define the type of variable:

c Constant: global parameters whose value should not be changed.

g Parameters whose value should only be set globally.

l Parameters whose value should only be set locally.

Each variable name is then build up in a similar way to that of a function, typically starting with the module\footnote{The module names are not used in case of generic scratch registers defined in the data type modules, \textit{e.g.}, the \texttt{int} module contains some scratch variables called \l_tmpa_int, \l_tmpb_int, and so on. In such a case adding the module name up front to denote the module and in the back to indicate the type, as in \l_int_tmpa_int would be very unreadable.} name and then a descriptive part. Variables end with a short identifier to show the variable type:

\texttt{bool} Either true or false.

\texttt{box} Box register.
clist Comma separated list.

coffin a “box with handles” — a higher-level data type for carrying out box alignment operations.

dim “Rigid” lengths.

fp floating-point values;

int Integer-valued count register.

prop Property list.

seq “Sequence”: a data-type used to implement lists (with access at both ends) and stacks.

skip “Rubber” lengths.

stream An input or output stream (for reading from or writing to, respectively).

tl Token list variables: placeholder for a token list.

1.1 Terminological inexactitude

A word of warning. In this document, and others referring to the expl3 programming modules, we often refer to “variables” and “functions” as if they were actual constructs from a real programming language. In truth, \TeX{} is a macro processor, and functions are simply macros that may or may not take arguments and expand to their replacement text. Many of the common variables are also macros, and if placed into the input stream will simply expand to their definition as well — a “function” with no arguments and a “token list variable” are in truth one and the same. On the other hand, some “variables” are actually registers that must be initialised and their values set and retrieved with specific functions.

The conventions of the expl3 code are designed to clearly separate the ideas of “macros that contain data” and “macros that contain code”, and a consistent wrapper is applied to all forms of “data” whether they be macros or actually registers. This means that sometimes we will use phrases like “the function returns a value”, when actually we just mean “the macro expands to something”. Similarly, the term “execute” might be used in place of “expand” or it might refer to the more specific case of “processing in \TeX{}‘s stomach” (if you are familiar with the \TeX{}book parlance).

If in doubt, please ask; chances are we’ve been hasty in writing certain definitions and need to be told to tighten up our terminology.

2 Documentation conventions

This document is typeset with the experimental \l3doc{} class; several conventions are used to help describe the features of the code. A number of conventions are used here to make the documentation clearer.
Each group of related functions is given in a box. For a function with a “user” name, this might read:

```
\ExplSyntaxOn
\ExplSyntaxOn ...
\ExplSyntaxOff
```

The textual description of how the function works would appear here. The syntax of the function is shown in mono-spaced text to the right of the box. In this example, the function takes no arguments and so the name of the function is simply reprinted.

For programming functions, which use _ and : in their name there are a few additional conventions: If two related functions are given with identical names but different argument specifiers, these are termed variants of each other, and the latter functions are printed in grey to show this more clearly. They will carry out the same function but will take different types of argument:

```
\seq_new:N \seq_new:N \langle sequence \rangle
\seq_new:c
```

When a number of variants are described, the arguments are usually illustrated only for the base function. Here, \langle sequence \rangle indicates that \seq_new:N expects the name of a sequence. From the argument specifier, \seq_new:c also expects a sequence name, but as a name rather than as a control sequence. Each argument given in the illustration should be described in the following text.

**Fully expandable functions** Some functions are fully expandable, which allows it to be used within an x-type argument (in plain TEX terms, inside an \edef), as well as within an f-type argument. These fully expandable functions are indicated in the documentation by a star:

```
\cs_to_str:N ∗ \cs_to_str:N \langle cs \rangle
```

As with other functions, some text should follow which explains how the function works. Usually, only the star will indicate that the function is expandable. In this case, the function expects a \langle cs \rangle, shorthand for a \langle control sequence \rangle.

**Restricted expandable functions** A few functions are fully expandable but cannot be fully expanded within an f-type argument. In this case a hollow star is used to indicate this:

```
\seq_map_function:NN ∗ \seq_map_function:NN \langle seq \rangle \langle function \rangle
```

**Conditional functions** Conditional (if) functions are normally defined in three variants, with T, F and TF argument specifiers. This allows them to be used for different “true”/“false” branches, depending on which outcome the conditional is being used to test. To indicate this without repetition, this information is given in a shortened form:
The underlining and italic of $\text{T} \text{F}$ indicates that $\text{T}$, $\text{F}$ and $\text{TF}$ are all available. Usually, the illustration will use the $\text{T}$ variant, and so both $\langle \text{true code} \rangle$ and $\langle \text{false code} \rangle$ will be shown. The two variant forms $\text{T}$ and $\text{F}$ take only $\langle \text{true code} \rangle$ and $\langle \text{false code} \rangle$, respectively. Here, the star also shows that this function is expandable. With some minor exceptions, all conditional functions in the expl3 modules should be defined in this way.

Variables, constants and so on are described in a similar manner:

A short piece of text will describe the variable: there is no syntax illustration in this case.

In some cases, the function is similar to one in LATEX2ε or plain LATEX. In these cases, the text will include an extra “\text{TEXhackers note}” section:

The normal description text.

\text{TEXhackers note}: Detail for the experienced LATEX or LATEX2ε programmer. In this case, it would point out that this function is the LATEX primitive \texttt{\texttt{string}}.

Changes to behaviour When new functions are added to expl3, the date of first inclusion is given in the documentation. Where the documented behaviour of a function changes after it is first introduced, the date of the update will also be given. This means that the programmer can be sure that any release of expl3 after the date given will contain the function of interest with expected behaviour as described. Note that changes to code internals, including bug fixes, are not recorded in this way unless they impact on the expected behaviour.

3 Formal language conventions which apply generally

As this is a formal reference guide for LATEX3 programming, the descriptions of functions are intended to be reasonably “complete”. However, there is also a need to avoid repetition. Formal ideas which apply to general classes of function are therefore summarised here.

For tests which have a TF argument specification, the test if evaluated to give a logically TRUE or FALSE result. Depending on this result, either the \langle true code \rangle or the \langle false code \rangle will be left in the input stream. In the case where the test is expandable, and a predicate \langle _p \rangle variant is available, the logical value determined by the test is left in the input stream: this will typically be part of a larger logical construct.
4 TeX concepts not supported by \TeX3

The TeX concept of an "\outer" macro is not supported at all by \TeX3. As such, the functions provided here may break when used on top of \TeX2\epsilon if \outer tokens are used in the arguments.
Part II

The \texttt{l3bootstrap} package

Bootstrap code

1 Using the \LaTeX{}X3 modules

The modules documented in \texttt{source3} are designed to be used on top of \LaTeX{}X2ε and are loaded all as one with the usual \texttt{\usepackage{expl3}} or \texttt{\RequirePackage{expl3}} instructions. These modules will also form the basis of the \LaTeX{}X3 format, but work in this area is incomplete and not included in this documentation at present.

As the modules use a coding syntax different from standard \LaTeX{}X2ε it provides a few functions for setting it up.

\begin{verbatim}
\ExplSyntaxOn
\ExplSyntaxOn \langle code \rangle \ExplSyntaxOff
\end{verbatim}

The \texttt{\ExplSyntaxOn} function switches to a category code régime in which spaces are ignored and in which the colon (:) and underscore (_) are treated as “letters”, thus allowing access to the names of code functions and variables. Within this environment, ~ is used to input a space. The \texttt{\ExplSyntaxOff} reverts to the document category code régime.

\begin{verbatim}
\ProvidesExplPackage \ProvidesExplClass \ProvidesExplFile
\end{verbatim}

\begin{verbatim}
\GetIdInfo $Id: ⟨SVN info field⟩ \{description} \end{verbatim}

Extracts all information from a SVN field. Spaces are not ignored in these fields. The information pieces are stored in separate control sequences with \texttt{\ExplFileName} for the part of the file name leading up to the period, \texttt{\ExplFileDate} for date, \texttt{\ExplFileVersion} for version and \texttt{\ExplFileDescription} for the description.

To summarize: Every single package using this syntax should identify itself using one of the above methods. Special care is taken so that every package or class file loaded with \texttt{\RequirePackage} or alike are loaded with usual \LaTeX{}X2ε category codes and the \LaTeX{}X3 category code scheme is reloaded when needed afterwards. See implementation for details. If you use the \texttt{\GetIdInfo} command you can use the information when loading a package with

\begin{verbatim}
\ProvidesExplPackage{\ExplFileName}
\ExplFileDate{\ExplFileVersion}{\ExplFileDescription}
\end{verbatim}
1.1 Internal functions and variables

\l__kernel_expl_bool A boolean which records the current code syntax status: true if currently inside a code environment. This variable should only be set by \ExplSyntaxOn/\ExplSyntaxOff.
Part III

The \texttt{l3names} package

Namespace for primitives

1 Setting up the \LaTeXe\ programming language

This module is at the core of the \LaTeXe\ programming language. It performs the following tasks:

- defines new names for all \TeX\ primitives;
- switches to the category code régime for programming;
- provides support settings for building the code as a \TeX\ format.

This module is entirely dedicated to primitives, which should not be used directly within \LaTeXe\ code (outside of “kernel-level” code). As such, the primitives are not documented here: \textit{The \TeX\book}, \textit{\TeX\ by Topic} and the manuals for pdf\TeX, X\TeX and Lua\TeX should be consulted for details of the primitives. These are named based on the engine which first introduced them:

\begin{itemize}
  \item \texttt{\tex...} Introduced by \TeX\ itself;
  \item \texttt{\etex...} Introduced by the $\varepsilon$-\TeX\ extensions;
  \item \texttt{\pdftex...} Introduced by pdf\TeX;
  \item \texttt{\xetex...} Introduced by X\TeX;
  \item \texttt{\luatex...} Introduced by Lua\TeX.
\end{itemize}
Part IV

The l3basics package

Basic definitions

As the name suggest this package holds some basic definitions which are needed by most or all other packages in this set.

Here we describe those functions that are used all over the place. With that we mean functions dealing with the construction and testing of control sequences. Furthermore the basic parts of conditional processing are covered: conditional processing dealing with specific data types is described in the modules specific for the respective data types.

1 No operation functions

\prg_do_nothing: * \prg_do_nothing:

An expandable function which does nothing at all: leaves nothing in the input stream after a single expansion.

\scan_stop: \scan_stop:

A non-expandable function which does nothing. Does not vanish on expansion but produces no typeset output.

2 Grouping material

\group_begin: \group_end:

These functions begin and end a group for definition purposes. Assignments are local to groups unless carried out in a global manner. (A small number of exceptions to this rule will be noted as necessary elsewhere in this document.) Each \group_begin: must be matched by a \group_end:, although this does not have to occur within the same function. Indeed, it is often necessary to start a group within one function and finish it within another, for example when seeking to use non-standard category codes.

\group_insert_after:N \group_insert_after:N \langle token \rangle

Adds \langle token \rangle to the list of \langle tokens \rangle to be inserted when the current group level ends. The list of \langle tokens \rangle to be inserted will be empty at the beginning of a group: multiple applications of \group_insert_after:N may be used to build the inserted list one \langle token \rangle at a time. The current group level may be closed by a \group_end: function or by a token with category code 2 (close-group). The later will be a } if standard category codes apply.
3 Control sequences and functions

As \TeX{} is a macro language, creating new functions means creating macros. At point of use, a function is replaced by the replacement text ("code") in which each parameter in the code (#1, #2, etc.) is replaced the appropriate arguments absorbed by the function. In the following, \textit{(code)} is therefore used as a shorthand for "replacement text".

Functions which are not “protected” will be fully expanded inside an \texttt{x} expansion. In contrast, “protected” functions are not expanded within \texttt{x} expansions.

3.1 Defining functions

Functions can be created with no requirement that they are declared first (in contrast to variables, which must always be declared). Declaring a function before setting up the code means that the name chosen will be checked and an error raised if it is already in use. The name of a function can be checked at the point of definition using the \texttt{\textbackslash cs\_new\ldots} functions: this is recommended for all functions which are defined for the first time.

There are three ways to define new functions. All classes define a function to expand to the substitution text. Within the substitution text the actual parameters are substituted for the formal parameters (#1, #2, \ldots).

- \texttt{new} Create a new function with the \texttt{new} scope, such as \texttt{\cs\_new\_Npn}. The definition is global and will result in an error if it is already defined.
- \texttt{set} Create a new function with the \texttt{set} scope, such as \texttt{\cs\_set\_Npn}. The definition is restricted to the current \TeX{} group and will not result in an error if the function is already defined.
- \texttt{gset} Create a new function with the \texttt{gset} scope, such as \texttt{\cs\_gset\_Npn}. The definition is global and will not result in an error if the function is already defined.

Within each set of scope there are different ways to define a function. The differences depend on restrictions on the actual parameters and the expandability of the resulting function.

- \texttt{nopar} Create a new function with the \texttt{nopar} restriction, such as \texttt{\cs\_set\_nopar\_Npn}. The parameter may not contain \texttt{\par} tokens.
- \texttt{protected} Create a new function with the \texttt{protected} restriction, such as \texttt{\cs\_set\_protected\_Npn}. The parameter may contain \texttt{\par} tokens but the function will not expand within an \texttt{x}-type expansion.

Finally, the functions in Subsections 3.2 and 3.3 are primarily meant to define base \textit{functions} only. Base functions can only have the following argument specifiers:

- \texttt{N} and \texttt{n} No manipulation.
- \texttt{T} and \texttt{F} Functionally equivalent to \texttt{n} (you are actually encouraged to use the family of \texttt{\prg\_new\_conditional}: functions described in Section 1).
- \texttt{p} and \texttt{w} These are special cases.
The \texttt{\cs_new}: functions below (and friends) do not stop you from using other argument specifiers in your function names, but they do not handle expansion for you. You should define the base function and then use \texttt{\cs_generate_variant:Nn} to generate custom variants as described in Section 2.

### 3.2 Defining new functions using parameter text

\begin{verbatim}
\cs_new:Npn \langle function \rangle \langle parameters \rangle \{\langle code \rangle\}
\end{verbatim}

Creates \texttt{\langle function \rangle} to expand to \texttt{\langle code \rangle} as replacement text. Within the \texttt{\langle code \rangle}, the \texttt{\langle parameters \rangle} (\texttt{\#1}, \texttt{\#2}, \texttt{etc.}) will be replaced by those absorbed by the function. The definition is global and an error will result if the \texttt{\langle function \rangle} is already defined.

\begin{verbatim}
\cs_new_nopar:Npn \langle function \rangle \langle parameters \rangle \{\langle code \rangle\}
\end{verbatim}

Creates \texttt{\langle function \rangle} to expand to \texttt{\langle code \rangle} as replacement text. Within the \texttt{\langle code \rangle}, the \texttt{\langle parameters \rangle} (\texttt{\#1}, \texttt{\#2}, \texttt{etc.}) will be replaced by those absorbed by the function. When the \texttt{\langle function \rangle} is used the \texttt{\langle parameters \rangle} absorbed cannot contain \texttt{\par} tokens. The definition is global and an error will result if the \texttt{\langle function \rangle} is already defined.

\begin{verbatim}
\cs_set:Npn \langle function \rangle \langle parameters \rangle \{\langle code \rangle\}
\end{verbatim}

Sets \texttt{\langle function \rangle} to expand to \texttt{\langle code \rangle} as replacement text. Within the \texttt{\langle code \rangle}, the \texttt{\langle parameters \rangle} (\texttt{\#1}, \texttt{\#2}, \texttt{etc.}) will be replaced by those absorbed by the function. The assignment of a meaning to the \texttt{\langle function \rangle} is restricted to the current \TeX{} group level.
\cs_set_nopar:Npn \langle function \rangle \langle parameters \rangle \{\langle code \rangle\}

Sets \langle function \rangle to expand to \langle code \rangle as replacement text. Within the \langle code \rangle, the \langle parameters \rangle (#1, #2, etc.) will be replaced by those absorbed by the function. When the \langle function \rangle is used the \langle parameters \rangle absorbed cannot contain \par tokens. The assignment of a meaning to the \langle function \rangle is restricted to the current \TeX group level.

\cs_set_protected:Npn \langle function \rangle \langle parameters \rangle \{\langle code \rangle\}

Sets \langle function \rangle to expand to \langle code \rangle as replacement text. Within the \langle code \rangle, the \langle parameters \rangle (#1, #2, etc.) will be replaced by those absorbed by the function. The assignment of a meaning to the \langle function \rangle is restricted to the current \TeX group level. The \langle function \rangle will not expand within an x-type argument.

\cs_gset:Npn \langle function \rangle \langle parameters \rangle \{\langle code \rangle\}

Globally sets \langle function \rangle to expand to \langle code \rangle as replacement text. Within the \langle code \rangle, the \langle parameters \rangle (#1, #2, etc.) will be replaced by those absorbed by the function. The assignment of a meaning to the \langle function \rangle is not restricted to the current \TeX group level: the assignment is global.

\cs_gset_protected:Npn \langle function \rangle \langle parameters \rangle \{\langle code \rangle\}

Globally sets \langle function \rangle to expand to \langle code \rangle as replacement text. Within the \langle code \rangle, the \langle parameters \rangle (#1, #2, etc.) will be replaced by those absorbed by the function. When the \langle function \rangle is used the \langle parameters \rangle absorbed cannot contain \par tokens. The assignment of a meaning to the \langle function \rangle is not restricted to the current \TeX group level: the assignment is global. The \langle function \rangle will not expand within an x-type argument.
Globally sets \{function\} to expand to \{code\} as replacement text. Within the \{code\}, the \{parameters\} (#1, #2, etc.) will be replaced by those absorbed by the function. When the \{function\} is used the \{parameters\} absorbed cannot contain \texttt{\par} tokens. The assignment of a meaning to the \{function\} is not restricted to the current \TeX{} group level; the assignment is global. The \{function\} will not expand within an x-type argument.

3.3 Defining new functions using the signature

\begin{verbatim}
\cs_new:Nn \function \{\code\}
\end{verbatim}

Creates \{function\} to expand to \{code\} as replacement text. Within the \{code\}, the number of \{parameters\} is detected automatically from the function signature. These \{parameters\} (#1, #2, etc.) will be replaced by those absorbed by the function. The definition is global and an error will result if the \{function\} is already defined.

\begin{verbatim}
\cs_new:cpn \function \{\code\}
\end{verbatim}

Creates \{function\} to expand to \{code\} as replacement text. Within the \{code\}, the number of \{parameters\} is detected automatically from the function signature. These \{parameters\} (#1, #2, etc.) will be replaced by those absorbed by the function. When the \{function\} is used the \{parameters\} absorbed cannot contain \texttt{\par} tokens. The definition is global and an error will result if the \{function\} is already defined.

\begin{verbatim}
\cs_new_protected:Nn \function \{\code\}
\end{verbatim}

Creates \{function\} to expand to \{code\} as replacement text. Within the \{code\}, the number of \{parameters\} is detected automatically from the function signature. These \{parameters\} (#1, #2, etc.) will be replaced by those absorbed by the function. The \{function\} will not expand within an x-type argument. The definition is global and an error will result if the \{function\} is already defined.
\cs_set:Nn \cs_set:(cnNxcx) \cs_set:Nn \{function\} \{\<code\}\}

Sets \texttt{\textbackslash function} to expand to \texttt{\<code\>} as replacement text. Within the \texttt{\<code\>}, the number of \texttt{\<parameters\>} is detected automatically from the function signature. These \texttt{\<parameters\>} \texttt{(#1, #2, \textit{etc.})} will be replaced by those absorbed by the function. The assignment of a meaning to the \texttt{\<function\>} is restricted to the current \TeX\ group level.

\cs_set:nopar:Nn \cs_set:nopar:(cnNxcx) \cs_set:nopar:Nn \{function\} \{\<code\}\}

Sets \texttt{\textbackslash function} to expand to \texttt{\<code\>} as replacement text. Within the \texttt{\<code\>}, the number of \texttt{\<parameters\>} is detected automatically from the function signature. These \texttt{\<parameters\>} \texttt{(#1, #2, \textit{etc.})} will be replaced by those absorbed by the function. When the \texttt{\<function\>} is used the \texttt{\<parameters\>} absorbed cannot contain \texttt{\textbackslash par} tokens. The assignment of a meaning to the \texttt{\<function\>} is restricted to the current \TeX\ group level.

\cs_set_protected:Nn \cs_set_protected:(cnNxcx) \cs_set_protected:Nn \{function\} \{\<code\}\}

Sets \texttt{\textbackslash function} to expand to \texttt{\<code\>} as replacement text. Within the \texttt{\<code\>}, the number of \texttt{\<parameters\>} is detected automatically from the function signature. These \texttt{\<parameters\>} \texttt{(#1, #2, \textit{etc.})} will be replaced by those absorbed by the function. The \texttt{\<function\>} will not expand within an \texttt{x-type} argument. The assignment of a meaning to the \texttt{\<function\>} is restricted to the current \TeX\ group level.

\cs_set_protected:nopar:Nn \cs_set_protected:nopar:(cnNxcx) \cs_set_protected:nopar:Nn \{function\} \{\<code\}\}

Sets \texttt{\textbackslash function} to expand to \texttt{\<code\>} as replacement text. Within the \texttt{\<code\>}, the number of \texttt{\<parameters\>} is detected automatically from the function signature. These \texttt{\<parameters\>} \texttt{(#1, #2, \textit{etc.})} will be replaced by those absorbed by the function. When the \texttt{\<function\>} is used the \texttt{\<parameters\>} absorbed cannot contain \texttt{\textbackslash par} tokens. The \texttt{\<function\>} will not expand within an \texttt{x-type} argument. The assignment of a meaning to the \texttt{\<function\>} is restricted to the current \TeX\ group level.

\cs_gset:Nn \cs_gset:(cnNxcx) \cs_gset:Nn \{function\} \{\<code\}\}

Sets \texttt{\<function\>} to expand to \texttt{\<code\>} as replacement text. Within the \texttt{\<code\>}, the number of \texttt{\<parameters\>} is detected automatically from the function signature. These \texttt{\<parameters\>} \texttt{(#1, #2, \textit{etc.})} will be replaced by those absorbed by the function. The assignment of a meaning to the \texttt{\<function\>} is global.

\cs_gset:nopar:Nn \cs_gset:nopar:(cnNxcx) \cs_gset:nopar:Nn \{function\} \{\<code\}\}

Sets \texttt{\<function\>} to expand to \texttt{\<code\>} as replacement text. Within the \texttt{\<code\>}, the number of \texttt{\<parameters\>} is detected automatically from the function signature. These \texttt{\<parameters\>} \texttt{(#1, #2, \textit{etc.})} will be replaced by those absorbed by the function. When the \texttt{\<function\>} is used the \texttt{\<parameters\>} absorbed cannot contain \texttt{\textbackslash par} tokens. The assignment of a meaning to the \texttt{\<function\>} is global.
\cs_gset_protected:Nn \cs_gset_protected:Nn (function) \{\langle code\rangle\}
\cs_gset_protected:(cn|Nx|cx)

Sets \textit{(function)} to expand to \textit{(code)} as replacement text. Within the \textit{(code)}, the number of \textit{(parameters)} is detected automatically from the function signature. These \textit{(parameters)} (\#1, \#2, \textit{etc.}) will be replaced by those absorbed by the function. The \textit{(function)} will not expand within an \texttt{x}-type argument. The assignment of a meaning to the \textit{(function)} is global.

\cs_gset_protected_nopar:Nn \cs_gset_protected_nopar:Nn (function) \{\langle code\rangle\}
\cs_gset_protected_nopar:(cn|Nx|cx)

Sets \textit{(function)} to expand to \textit{(code)} as replacement text. Within the \textit{(code)}, the number of \textit{(parameters)} is detected automatically from the function signature. These \textit{(parameters)} (\#1, \#2, \textit{etc.}) will be replaced by those absorbed by the function. When the \textit{(function)} is used the \textit{(parameters)} absorbed cannot contain \texttt{\par} tokens. The \textit{(function)} will not expand within an \texttt{x}-type argument. The assignment of a meaning to the \textit{(function)} is global.

\cs_generate_from_arg_count:NNnn \cs_generate_from_arg_count:NNnn (creator) (number) \{\langle code\rangle\}
\cs_generate_from_arg_count:(cNnn|Ncnn)(cNnn|Ncnn)

Updated: 2012-01-14

Uses the \textit{(creator)} function (which should have signature \texttt{Npn}, for example \texttt{\cs_new:Npn}) to define a \textit{(function)} which takes \textit{(number)} arguments and has \textit{(code)} as replacement text. The \textit{(number)} of arguments is an integer expression, evaluated as detailed for \texttt{\int_eval:n}.

### 3.4 Copying control sequences

Control sequences (not just functions as defined above) can be set to have the same meaning using the functions described here. Making two control sequences equivalent means that the second control sequence is a \textit{copy} of the first (rather than a pointer to it). Thus the old and new control sequence are not tied together: changes to one are not reflected in the other.

In the following text “\textit{cs}” is used as an abbreviation for “control sequence”.

\cs_new_eq:NN \cs_new_eq:NN \{cs\} \{cs2\}
\cs_new_eq:NN \cs_new_eq:NN \{cs\} \{token\}

Globally creates \textit{(control sequence)} and sets it to have the same meaning as \textit{(control sequence2)} or \textit{(token)}. The second control sequence may subsequently be altered without affecting the copy.
\cs_set_eq:NN \cs_set_eq:(NcN[cc]) \cs_set_eq:NN \langle control sequence \rangle \langle control sequence \rangle
\cs_set_eq:NN \langle control sequence \rangle \langle token \rangle

Sets \langle control sequence_1 \rangle to have the same meaning as \langle control sequence_2 \rangle (or \langle token \rangle). The second control sequence may subsequently be altered without affecting the copy. The assignment of a meaning to the \langle control sequence \rangle is restricted to the current \TeX group level.

\cs_gset_eq:NN \cs_gset_eq:(NcN[cc]) \cs_gset_eq:NN \langle control sequence \rangle \langle control sequence \rangle
\cs_gset_eq:NN \langle control sequence \rangle \langle token \rangle

Globally sets \langle control sequence_1 \rangle to have the same meaning as \langle control sequence_2 \rangle (or \langle token \rangle). The second control sequence may subsequently be altered without affecting the copy. The assignment of a meaning to the \langle control sequence \rangle is not restricted to the current \TeX group level: the assignment is global.

3.5 Deleting control sequences

There are occasions where control sequences need to be deleted. This is handled in a very simple manner.

\cs_undefine:N \cs_undefine:c

Sets \langle control sequence \rangle to be globally undefined.

3.6 Showing control sequences

\cs_meaning:N \cs_meaning:c

This function expands to the meaning of the \langle control sequence \rangle control sequence. This will show the \langle replacement text \rangle for a macro.

\TeXhacks{This is \TeX’s \texttt{\meaning} primitive. The \texttt{c} variant correctly reports undefined arguments.}

\cs_show:N \cs_show:c

Displays the definition of the \langle control sequence \rangle on the terminal.

\TeXhacks{This is similar to the \TeX primitive \texttt{\show}, wrapped to a fixed number of characters per line.}
3.7 Converting to and from control sequences

\use:c \{(control sequence name)\}

Converts the given \{(control sequence name)\} into a single control sequence token. This process requires two expansions. The content for \{(control sequence name)\} may be literal material or from other expandable functions. The \{(control sequence name)\} must, when fully expanded, consist of character tokens which are not active: typically, they will be of category code 10 (space), 11 (letter) or 12 (other), or a mixture of these.

As an example of the \use:c function, both

\use:c { a b c }

and

\tl_new:N \l_my_tl
\tl_set:Nn \l_my_tl { a b c }
\use:c { \tl_use:N \l_my_tl }

would be equivalent to

\abc

after two expansions of \use:c.

\cs_if_exist_use:N \cs_if_exist_use:c

Tests whether the \{(control sequence)\} is currently defined (whether as a function or another control sequence type), and if it does inserts the \{(control sequence)\} into the input stream.

\cs_if_exist_use:NTF \cs_if_exist_use:TF

Tests whether the \{(control sequence)\} is currently defined (whether as a function or another control sequence type), and if it does inserts the \{(control sequence)\} into the input stream followed by the \{(true code)\}.

\cs:w \cs_end:

Converts the given \{(control sequence name)\} into a single control sequence token. This process requires one expansion. The content for \{(control sequence name)\} may be literal material or from other expandable functions. The \{(control sequence name)\} must, when fully expanded, consist of character tokens which are not active: typically, they will be of category code 10 (space), 11 (letter) or 12 (other), or a mixture of these.

\TeX{}hackers note: These are the \TeX{} primitives \csname and \endcsname.

As an example of the \cs:w and \cs_end: functions, both

\cs:w a b c \cs_end:

and
\tl_new:N \l_my_tl
\tl_set:Nn \l_my_tl { a b c }
\cs:w \tl_use:N \l_my_tl \cs_end:

would be equivalent to

\abc

after one expansion of \cs:w.

\cs_to_str:N \cs_to_str:N (control sequence)

Converts the given \textit{(control sequence)} into a series of characters with category code 12 (other), except spaces, of category code 10. The sequence will not include the current escape token, \textit{cf.} \textbackslash token_to_str:N. Full expansion of this function requires exactly 2 expansion steps, and so an x-type expansion, or two o-type expansions will be required to convert the \textit{(control sequence)} to a sequence of characters in the input stream. In most cases, an f-expansion will be correct as well, but this loses a space at the start of the result.

4 Using or removing tokens and arguments

Tokens in the input can be read and used or read and discarded. If one or more tokens are wrapped in braces then in absorbing them the outer set will be removed. At the same time, the category code of each token is set when the token is read by a function (if it is read more than once, the category code is determined by the situation in force when first function absorbs the token).

\use:n \use:n \use:nn \use:nnn \use:nnnn

As illustrated, these functions will absorb between one and four arguments, as indicated by the argument specifier. The braces surrounding each argument will be removed leaving the remaining tokens in the input stream. The category code of these tokens will also be fixed by this process (if it has not already been by some other absorption). All of these functions require only a single expansion to operate, so that one expansion of

\use:nn { abc } { { def } }

will result in the input stream containing

abc { def }

\textit{i.e.} only the outer braces will be removed.
These functions absorb two arguments from the input stream. The function \use_i:nn discards the second argument, and leaves the content of the first argument in the input stream. \use_ii:nn discards the first argument and leaves the content of the second argument in the input stream. The category code of these tokens will also be fixed (if it has not already been by some other absorption). A single expansion is needed for the functions to take effect.

These functions absorb three arguments from the input stream. The function \use_i:nn discards the second and third arguments, and leaves the content of the first argument in the input stream. \use_ii:nn and \use_iii:nn work similarly, leaving the content of second or third arguments in the input stream, respectively. The category code of these tokens will also be fixed (if it has not already been by some other absorption). A single expansion is needed for the functions to take effect.

These functions absorb four arguments from the input stream. The function \use_-i:nnn discards the second, third and fourth arguments, and leaves the content of the first argument in the input stream. \use_ii:nnn, \use_iii:nnn and \use_iv:nnn work similarly, leaving the content of second, third or fourth arguments in the input stream, respectively. The category code of these tokens will also be fixed (if it has not already been by some other absorption). A single expansion is needed for the functions to take effect.

This functions will absorb three arguments and leave the content of the first and second in the input stream. The category code of these tokens will also be fixed (if it has not already been by some other absorption). A single expansion is needed for the functions to take effect. An example:

\use_i_ii:nnn { abc } { { def } } { ghi }

will result in the input stream containing

abc { def }

i.e. the outer braces will be removed and the third group will be removed.
These functions absorb between one and nine groups from the input stream, leaving nothing on the resulting input stream. These functions work after a single expansion. One or more of the \( n \) arguments may be an unbraced single token (\textit{i.e.} an \( N \) argument).

\[ \text{Absorb the } \langle \text{balanced text} \rangle \text{ form the input stream delimited by the marker given in the function name, leaving } \langle \text{inserted tokens} \rangle \text{ in the input stream for further processing.} \]

\[ \text{Absorb the } \langle \text{balanced text} \rangle \text{ form the input stream delimited by the marker given in the function name, leaving } \langle \text{inserted tokens} \rangle \text{ in the input stream for further processing.} \]

\textbf{5 Predicates and conditionals}

\LaTeX{} has three concepts for conditional flow processing:

\textbf{Branching conditionals} Functions that carry out a test and then execute, depending on its result, either the code supplied as the (\textit{true code}) or the (\textit{false code}). These arguments are denoted with T and F, respectively. An example would be

\[ \\text{\cs_if_free:cTF \{abc\} \{(true code)\} \{(false code)\}} \]
a function that will turn the first argument into a control sequence (since it’s marked as \texttt{c}) then checks whether this control sequence is still free and then depending on the result carry out the code in the second argument (true case) or in the third argument (false case).

These type of functions are known as “conditionals”; whenever a \texttt{TF} function is defined it will usually be accompanied by \texttt{T} and \texttt{F} functions as well. These are provided for convenience when the branch only needs to go a single way. Package writers are free to choose which types to define but the kernel definitions will always provide all three versions.

Important to note is that these branching conditionals with \langle \texttt{true code} \rangle and/or \langle \texttt{false code} \rangle are always defined in a way that the code of the chosen alternative can operate on following tokens in the input stream.

These conditional functions may or may not be fully expandable, but if they are expandable they will be accompanied by a “predicate” for the same test as described below.

\textbf{Predicates} “Predicates” are functions that return a special type of boolean value which can be tested by the boolean expression parser. All functions of this type are expandable and have names that end with \_p in the description part. For example,

\begin{verbatim}
\cs_if_free_p:N
\end{verbatim}

would be a predicate function for the same type of test as the conditional described above. It would return “true” if its argument (a single token denoted by \texttt{N}) is still free for definition. It would be used in constructions like

\begin{verbatim}
\bool_if:nTF { \cs_if_free_p:N \l_tmpz_tl || \cs_if_free_p:N \g_tmpz_tl } {\langle true code\rangle} {\langle false code\rangle}
\end{verbatim}

For each predicate defined, a “branching conditional” will also exist that behaves like a conditional described above.

\textbf{Primitive conditionals} There is a third variety of conditional, which is the original concept used in plain \TeX and \LaTeX. Their use is discouraged in expl3 (although still used in low-level definitions) because they are more fragile and in many cases require more expansion control (hence more code) than the two types of conditionals described above.

\begin{verbatim}
c_true_bool
c_false_bool
\end{verbatim}

Constants that represent \texttt{true} and \texttt{false}, respectively. Used to implement predicates.
5.1 Tests on control sequences

\cs_if_eq_p:NN \langle cs_1 \rangle \{ \langle cs_2 \rangle \} \{true code\} \{false code\}
\cs_if_eq:NNF \langle cs_1 \rangle \{ \langle cs_2 \rangle \} \{true code\} \{false code\}

Comparing the definition of two \langle control sequences \rangle and is logically true if they have exactly the same definition when examined with \cs_show:N.

\cs_if_exist_p:N \langle control sequence \rangle
\cs_if_exist:NTF \langle control sequence \rangle \{true code\} \{false code\}

Tests whether the \langle control sequence \rangle is currently defined (whether as a function or another control sequence type). Any valid definition of \langle control sequence \rangle will evaluate as true.

\cs_if_free_p:N \langle control sequence \rangle
\cs_if_free:NTF \langle control sequence \rangle \{true code\} \{false code\}

Tests whether the \langle control sequence \rangle is currently free to be defined. This test will be false if the \langle control sequence \rangle currently exists (as defined by \cs_if_exist:N).

5.2 Engine-specific conditionals

\luatex_if_engine:TF \{true code\} \{false code\}

Detected the document is being compiled using LuaTEX.

\pdftex_if_engine:TF \{true code\} \{false code\}

Detected the document is being compiled using pdfTEX.

\xetex_if_engine:TF \{true code\} \{false code\}

Detected the document is being compiled using XeTEX.

5.3 Primitive conditionals

The \LaTeX engine itself provides many different conditionals. Some expand whatever comes after them and others don’t. Hence the names for these underlying functions will often contain a :w part but higher level functions are often available. See for instance \int_compare_p:nNn which is a wrapper for \if_int_compare:w.

Certain conditionals deal with specific data types like boxes and fonts and are described there. The ones described below are either the universal conditionals or deal with control sequences. We will prefix primitive conditionals with \if_.
\textbf{\texttt{\textbackslash{}if\_true}:} \texttt{\textbackslash{}if\_true: (true code) \texttt{\textbackslash{}else: (false code) \textbackslash{}fi:}}
\textbf{\texttt{\textbackslash{}if\_false}:} \texttt{\textbackslash{}if\_false: (true code) \texttt{\textbackslash{}else: (false code) \textbackslash{}fi:}}
\textbf{\texttt{\textbackslash{}else:}} \texttt{\textbackslash{}reverse\_if:N (primitive conditional)}
\textbf{\texttt{\textbackslash{}fi:}} \texttt{\textbackslash{}if\_true: always executes (true code), while \texttt{\textbackslash{}if\_false: always executes (false code). \texttt{\textbackslash{}reverse\_if:N} reverses any two-way primitive conditional. \texttt{\textbackslash{}else: and \textbackslash{}fi: delimit the branches of the conditional. The function \texttt{\textbackslash{}or:} is documented in \texttt{l3int} and used in case switches.}

\textbf{\texttt{\textbackslash{}if\_true:}} \texttt{\textbackslash{}reverse\_if:N (primitive conditional) \texttt{\textbackslash{}else:}} \texttt{\textbackslash{}reverse\_if:N is \texttt{\textbackslash{}unless}.}

\textbf{\texttt{\textbackslash{}if\_meaning:w \textbackslash{}if\_meaning:w (arg\textsubscript{1}) (arg\textsubscript{2}) (true code) \texttt{\textbackslash{}else: (false code) \textbackslash{}fi:}}}
\textbf{\texttt{\textbackslash{}if\_meaning:w}} \texttt{executes (true code) when (arg\textsubscript{1}) and (arg\textsubscript{2}) are the same, otherwise it executes (false code). (arg\textsubscript{1}) and (arg\textsubscript{2}) could be functions, variables, tokens; in all cases the \textit{unexpanded} definitions are compared.}

\textbf{\texttt{\textbackslash{}if\_meaning:w}} \texttt{is similar to \texttt{\textbackslash{}if\_meaning:w.}}

\textbf{\texttt{\textbackslash{}if\_charcode:w \textbackslash{}if\_charcode:w (token\textsubscript{1}) (token\textsubscript{2}) (true code) \texttt{\textbackslash{}else: (false code) \textbackslash{}fi:}}}
\textbf{\texttt{\textbackslash{}if\_charcode:w}} \texttt{tests if the character codes are identical. \texttt{\textbackslash{}if\_charcode:w} is an alternative name for \texttt{\textbackslash{}if\_w.}}

\textbf{\texttt{\textbackslash{}if\_catcode:w \textbackslash{}if\_catcode:w (token\textsubscript{1}) (token\textsubscript{2}) (true code) \texttt{\textbackslash{}else: (false code) \textbackslash{}fi:}}}
\textbf{\texttt{\textbackslash{}if\_catcode:w}} \texttt{tests if the category codes of the two tokens are the same whereas \texttt{\textbackslash{}if\_w} tests if the character codes are identical. \texttt{\textbackslash{}if\_catcode:w} is an alternative name for \texttt{\textbackslash{}if\_w.}}

\textbf{\texttt{\textbackslash{}if\_cs_exist:N \textbackslash{}if\_cs_exist:N (cs) (true code) \texttt{\textbackslash{}else: (false code) \textbackslash{}fi:}}}
\textbf{\texttt{\textbackslash{}if\_cs_exist:w \textbackslash{}if\_cs_exist:w (tokens) \texttt{cs\_end: (true code) \texttt{\textbackslash{}else: (false code) \textbackslash{}fi:}}}
\textbf{\texttt{Check if (cs) appears in the hash table or if the control sequence that can be formed from (tokens) appears in the hash table. The latter function does not turn the control sequence in question into \texttt{\textbackslash{}scan\_stop:}! This can be useful when dealing with control sequences which cannot be entered as a single token.}

\textbf{\texttt{\textbackslash{}if\_mode_horizontal: \textbackslash{}if\_mode_horizontal: (true code) \texttt{\textbackslash{}else: (false code) \textbackslash{}fi:}}}
\textbf{\texttt{\textbackslash{}if\_mode_horizontal:}} \texttt{execute (true code) if currently in horizontal mode, otherwise execute (false code). Similar for the other functions.}

\textbf{6 Internal kernel functions}

\textbf{\texttt{\_\_chk\_if\_exist\_cs:N \_\_chk\_if\_exist\_cs:N (cs)}}
\texttt{This function checks that (cs) exists according to the criteria for \texttt{\cs\_if\_exist\_p:N}, and if not raises a kernel-level error.}

24
This function checks that \( \text{cs} \) is free according to the criteria for \texttt{cs_if_free_p:N}, and if not raises a kernel-level error.

This function checks that \( \text{var} \) is defined according to the criteria for \texttt{cs_if_free_p:N}, and if not raises a kernel-level error. This function is only created if the package option \texttt{check-declarations} is active.

Splits the \( \text{function} \) into the \( \text{name} \) (i.e. the part before the colon) and the \( \text{signature} \) (i.e. after the colon). The \( \text{number} \) of tokens in the \( \text{signature} \) is then left in the input stream. If there was no \( \text{signature} \) then the result is the marker value \(-1\).

Splits the \( \text{function} \) into the \( \text{name} \) (i.e. the part before the colon) and the \( \text{signature} \) (i.e. after the colon). The \( \text{name} \) is then left in the input stream without the escape character present made up of tokens with category code \( 12 \) (other). The \( \text{processor} \) should be a function with argument specification \texttt{:nnN} (plus any trailing arguments needed).

Function used for various short-term usages, for instance defining functions whose definition involves tokens which are hard to insert normally (spaces, characters with category other).

Used to show the contents of a \TeX{} register at the terminal, formatted such that internal parts of the mechanism are not visible.
\__prg_case_end:nw * \__prg_case_end:nw \{(code)\} \q_mark \{(true code)\} \q_mark \{(false code)\} \q_stop

Used to terminate case statements (\texttt{int\_case:nnTF}, etc.) by removing trailing \texttt{\{tokens\}} and the end marker \texttt{\q\_stop}, inserting the \texttt{\{code\}} for the successful case (if one is found) and either the \texttt{true code} or \texttt{false code} for the overall outcome, as appropriate.
Part V
The \texttt{l3expan} package
\textbf{Argument expansion}

This module provides generic methods for expanding \TeX\ arguments in a systematic manner. The functions in this module all have prefix \texttt{exp}.

Not all possible variations are implemented for every base function. Instead only those that are used within the \LaTeX\ kernel or otherwise seem to be of general interest are implemented. Consult the module description to find out which functions are actually defined. The next section explains how to define missing variants.

1 Defining new variants

The definition of variant forms for base functions may be necessary when writing new functions or when applying a kernel function in a situation that we haven’t thought of before.

Internally preprocessing of arguments is done with functions from the \texttt{exp_} module. They all look alike, an example would be \texttt{\exp_args:NNo}. This function has three arguments, the first and the second are a single tokens, while the third argument should be given in braces. Applying \texttt{\exp_args:NNo} will expand the content of third argument once before any expansion of the first and second arguments. If \texttt{\seq_gpush:No} was not defined it could be coded in the following way:

\begin{verbatim}
\exp_args:NNo \seq_gpush:Nn
\g_file_name_stack \l_tmpa_tl
\end{verbatim}

In other words, the first argument to \texttt{\exp_args:NNo} is the base function and the other arguments are preprocessed and then passed to this base function. In the example the first argument to the base function should be a single token which is left unchanged while the second argument is expanded once. From this example we can also see how the variants are defined. They just expand into the appropriate \texttt{exp_} function followed by the desired base function, \textit{e.g.}

\begin{verbatim}
\cs_new_nopar:Npn \seq_gpush:No { \exp_args:NNo \seq_gpush:Nn }
\end{verbatim}

Providing variants in this way in style files is uncritical as the \texttt{\cs_new_nopar:Npn} function will silently accept definitions whenever the new definition is identical to an already given one. Therefore adding such definition to later releases of the kernel will not make such style files obsolete.

The steps above may be automated by using the function \texttt{\cs_generate_variant:NNn}, described next.
2 Methods for defining variants

\cs_generate_variant:Nn \parent_control_sequence \{ \variant_argument_specifiers \}

This function is used to define argument-specifier variants of the \parent_control_sequence for \LaTeX{} code-level macros. The \parent_control_sequence is first separated into the (base name) and \original_argument_specifier. The comma-separated list of \variant_argument_specifiers is then used to define variants of the \original_argument_specifier where these are not already defined. For each \variant given, a function is created which will expand its arguments as detailed and pass them to the \parent_control_sequence. So for example

\begin{verbatim}
\cs_set:Npn \foo:Nn #1#2 { code here }
\cs_generate_variant:Nn \foo:Nn { c }
\end{verbatim}

will create a new function \foo:cn which will expand its first argument into a control sequence name and pass the result to \foo:Nn. Similarly

\begin{verbatim}
\cs_generate_variant:Nn \foo:Nn \{ NV , cV \}
\end{verbatim}

would generate the functions \foo:NV and \foo:cV in the same way. The \cs_generate_variant:Nn function can only be applied if the \parent_control_sequence is already defined. If the \parent_control_sequence is protected then the new sequence will also be protected. The \variant is created globally, as is any \exp_args:N\variant function needed to carry out the expansion.

3 Introducing the variants

The available internal functions for argument expansion come in two flavours, some of them are faster then others. Therefore it is usually best to follow the following guidelines when defining new functions that are supposed to come with variant forms:

- Arguments that might need expansion should come first in the list of arguments to make processing faster.
- Arguments that should consist of single tokens should come first.
- Arguments that need full expansion (i.e., are denoted with \texttt{x}) should be avoided if possible as they can not be processed expandably, i.e., functions of this type will not work correctly in arguments that are themselves subject to \texttt{x} expansion.
- In general, unless in the last position, multi-token arguments \texttt{n}, \texttt{f}, and \texttt{e} will need special processing which is not fast. Therefore it is best to use the optimized functions, namely those that contain only \texttt{N}, \texttt{c}, \texttt{V}, and \texttt{v}, and, in the last position, \texttt{c}, \texttt{f}, with possible trailing \texttt{N} or \texttt{n}, which are not expanded.

The \texttt{V} type returns the value of a register, which can be one of \texttt{tl}, \texttt{num}, \texttt{int}, \texttt{skip}, \texttt{dim}, \texttt{toks}, or built-in \TeX{} registers. The \texttt{v} type is the same except it first creates a
control sequence out of its argument before returning the value. This recent addition to the argument specifiers may shake things up a bit as most places where \texttt{o} is used will be replaced by \texttt{V}. The documentation you are currently reading will therefore require a fair bit of re-writing.

In general, the programmer should not need to be concerned with expansion control. When simply using the content of a variable, functions with a \texttt{V} specifier should be used. For those referred to by \texttt{(cs)name}, the \texttt{v} specifier is available for the same purpose. Only when specific expansion steps are needed, such as when using delimited arguments, should the lower-level functions with \texttt{o} specifiers be employed.

The \texttt{f} type is so special that it deserves an example. Let’s pretend we want to set the control sequence whose name is given by \texttt{b \textbackslash l\texttt{t}mpa\texttt{t}l b} equal to the list of tokens \texttt{\aaa a}. Furthermore we want to store the execution of it in a \texttt{tl\texttt{v}ar}. In this example we assume \texttt{l\_tmpa\_tl} contains the text string \texttt{lur}. The straightforward approach is

\begin{verbatim}
\tl_set:No \l_tmpb_tl \{ \tl_set:cn \{ b \_l\_tmpa\_tl b \} \{ \aaa a \} \}
\end{verbatim}

Unfortunately this only puts \texttt{\exp_args:Nc \tl_set:Nn \{ b \_l\_tmpa\_tl b \} \{ \aaa a \}} into \texttt{l\_tmpb\_tl} and not \texttt{\tl_set:Nn \blurb \{ \aaa a \}} as we probably wanted. Using \texttt{\tl_set:Nx} is not an option as that will die horribly. Instead we can do a

\begin{verbatim}
\tl_set:Nf \l_tmpb_tl \{ \tl_set:cn \{ b \_l\_tmpa\_tl b \} \{ \aaa a \} \}
\end{verbatim}

which puts the desired result in \texttt{l\_tmpb\_tl}. It requires \texttt{\tl_set:Nf} to be defined as

\begin{verbatim}
\cs_set_nopar:Npn \tl_set:Nf \{ \exp_args:NNf \tl_set:Nn \}
\end{verbatim}

If you use this type of expansion in conditional processing then you should stick to using \texttt{TF} type functions only as it does not try to finish any \texttt{\if... \fi}: itself!

If is important to note that both \texttt{f}- and \texttt{o}-type expansion are concerned with the expansion of tokens from left to right in their arguments. In particular, \texttt{o}-type expansion applies to the first \texttt{token} in the argument it receives: it is conceptually similar to

\begin{verbatim}
\exp_after:wN <base function> \exp_after:wN \{ <argument> \}
\end{verbatim}

At the same time, \texttt{f}-type expansion stops at the emphfirst non-expandable token. This means for example that both

\begin{verbatim}
\tl_set:No \l_tmpa_tl \{ \{ \l_tmpa_tl \} \}
\end{verbatim}

and

\begin{verbatim}
\tl_set:Nf \l_tmpa_tl \{ \{ \l_tmpa_tl \} \}
\end{verbatim}

leave \texttt{l\_tmpa\_tl} unchanged: \{ is the first token in the argument and is non-expandable.

29
4 Manipulating the first argument

These functions are described in detail: expansion of multiple tokens follows the same
rules but is described in a shorter fashion.

:\exp_args:No \exp_args:No \langle \text{function} \rangle \{(\text{tokens})\} \ldots

This function absorbs two arguments (the \langle \text{function} \rangle name and the \langle \text{tokens} \rangle). The
\langle \text{tokens} \rangle are expanded once, and the result is inserted in braces into the input stream after
reinsertion of the \langle \text{function} \rangle. Thus the \langle \text{function} \rangle may take more than one argument: all others will be left unchanged.

:\exp_args:Nc \exp_args:Nc \exp_args:cc

This function absorbs two arguments (the \langle \text{function} \rangle name and the \langle \text{tokens} \rangle). The
\langle \text{tokens} \rangle are expanded until only characters remain, and are then turned into a control sequence. (An internal error will occur if such a conversion is not possible). The result is inserted into the input stream after reinserion of the \langle \text{function} \rangle. Thus the \langle \text{function} \rangle may take more than one argument: all others will be left unchanged.

The :cc variant constructs the \langle \text{function} \rangle name in the same manner as described for the
\langle \text{tokens} \rangle.

:\exp_args:NV \exp_args:NV \langle \text{function} \rangle \langle \text{variable} \rangle

This function absorbs two arguments (the names of the \langle \text{function} \rangle and the \langle \text{variable} \rangle). The
content of the \langle \text{variable} \rangle are recovered and placed inside braces into the input stream after reinserion of the \langle \text{function} \rangle. Thus the \langle \text{function} \rangle may take more than one argument: all others will be left unchanged.

:\exp_args:Nv \exp_args:Nv \langle \text{function} \rangle \{(\text{tokens})\}

This function absorbs two arguments (the \langle \text{function} \rangle name and the \langle \text{tokens} \rangle). The
\langle \text{tokens} \rangle are expanded until only characters remain, and are then turned into a control sequence. (An internal error will occur if such a conversion is not possible). This control sequence should be the name of a \langle \text{variable} \rangle. The content of the \langle \text{variable} \rangle are recovered and placed inside braces into the input stream after reinserion of the \langle \text{function} \rangle. Thus the \langle \text{function} \rangle may take more than one argument: all others will be left unchanged.

:\exp_args:Nf \exp_args:Nf \langle \text{function} \rangle \{(\text{tokens})\}

This function absorbs two arguments (the \langle \text{function} \rangle name and the \langle \text{tokens} \rangle). The
\langle \text{tokens} \rangle are fully expanded until the first non-expandable token or space is found, and
the result is inserted in braces into the input stream after reinserion of the \langle \text{function} \rangle. Thus the \langle \text{function} \rangle may take more than one argument: all others will be left unchanged.
This function absorbs two arguments (the \textit{function} name and the \textit{tokens}) and exhaustively expands the \textit{tokens} second. The result is inserted in braces into the input stream \textit{after} reinsertion of the \textit{function}. Thus the \textit{function} may take more than one argument: all others will be left unchanged.

5 Manipulating two arguments

\begin{center}
\begin{tabular}{l}
\texttt{\exp_args:Nx} \texttt{\exp_args:Nx (function) \{tokens\}} \\
\end{tabular}
\end{center}

These optimized functions absorb three arguments and expand the second and third as detailed by their argument specifier. The first argument of the function is then the next item on the input stream, followed by the expansion of the second and third arguments.

\begin{center}
\begin{tabular}{l}
\texttt{\exp_args:NNc (token1) (token2) \{tokens\}} \\
\texttt{\exp_args:NVV} \\
\texttt{\exp_args:Nnc} \\
\texttt{\exp_args:Nco} \\
\texttt{\exp_args:Ncf} \\
\end{tabular}
\end{center}

Updated: 2012-01-14

These functions absorb three arguments and expand the second and third as detailed by their argument specifier. The first argument of the function is then the next item on the input stream, followed by the expansion of the second and third arguments. These functions need special (slower) processing.

\begin{center}
\begin{tabular}{l}
\texttt{\exp_args:NNx} \\
\texttt{\exp_args:Ncx} \\
\texttt{\exp_args:Nnx} \\
\texttt{\exp_args:Nox} \\
\texttt{\exp_args:Nxo} \\
\texttt{\exp_args:Nxx} \\
\end{tabular}
\end{center}

These functions absorb three arguments and expand the second and third as detailed by their argument specifier. The first argument of the function is then the next item on the input stream, followed by the expansion of the second and third arguments. These functions are not expandable.
6 Manipulating three arguments

\exp_args:NNNo \exp_args:NNNo \langle token_1 \rangle \langle token_2 \rangle \langle token_3 \rangle \{\langle tokens \rangle\}
\exp_args:(NNNV|NcNo|Ncco) *
\exp_args:Nccc *
\exp_args:NcNc *

These optimized functions absorb four arguments and expand the second, third and fourth as detailed by their argument specifier. The first argument of the function is then the next item on the input stream, followed by the expansion of the second argument, etc.

\exp_args:NNNo \exp_args:NNNo \langle token_1 \rangle \langle token_2 \rangle \langle token_3 \rangle \{\langle tokens \rangle\}
\exp_args:(NNNV|NcNo|Ncco) *
\exp_args:Nccc *
\exp_args:NcNc *

These functions absorb four arguments and expand the second, third and fourth as detailed by their argument specifier. The first argument of the function is then the next item on the input stream, followed by the expansion of the second argument, etc. These functions need special (slower) processing.

\exp_args:NNNo \exp_args:NNNo \langle token_1 \rangle \langle token_2 \rangle \langle token_3 \rangle \{\langle tokens \rangle\}
\exp_args:(NNNV|NcNo|Ncco) *
\exp_args:Nccc *
\exp_args:NcNc *

These functions absorb four arguments and expand the second, third and fourth as detailed by their argument specifier. The first argument of the function is then the next item on the input stream, followed by the expansion of the second argument, etc.
7 Unbraced expansion

\exp_last_unbraced:Nf \exp_last_unbraced:Nno \exp_last_unbraced:Noo \exp_last_unbraced:Nfo
\exp_last_unbraced:Nco \exp_last_unbraced:NnNo \exp_last_unbraced:NNNV \exp_last_unbraced:NNNo

These functions absorb the number of arguments given by their specification, carry out the expansion indicated and leave the results in the input stream, with the last argument not surrounded by the usual braces. Of these, the :Nno, :Noo, and :Nfo variants need special (slower) processing.

\TeXhackers note: As an optimization, the last argument is unbraced by some of those functions before expansion. This can cause problems if the argument is empty: for instance, \exp_last_unbraced:Nf \mypkg_foo:w { } \q_stop leads to an infinite loop, as the quark is f-expanded.

\exp_last_unbraced:Nx
\exp_last_unbraced:N (function) {\langle tokens\rangle}

This function fully expands the \langle tokens\rangle and leaves the result in the input stream after reinsertion of \langle function\rangle. This function is not expandable.

\exp_last_two_unbraced:Noo
\exp_last_two_unbraced:Noo \langle token\rangle \langle tokens_1\rangle \langle tokens_2\rangle

This function absorbs three arguments and expand the second and third once. The first argument of the function is then the next item on the input stream, followed by the expansion of the second and third arguments, which are not wrapped in braces. This function needs special (slower) processing.

\exp_after:wN
\exp_after:wN \langle token_1\rangle \langle token_2\rangle

Carries out a single expansion of \langle token_2\rangle (which may consume arguments) prior to the expansion of \langle token_1\rangle. If \langle token_2\rangle is a \TeX primitive, it will be executed rather than expanded, while if \langle token_2\rangle has not expansion (for example, if it is a character) then it will be left unchanged. It is important to notice that \langle token_1\rangle may be any single token, including group-opening and -closing tokens (\{ or \} assuming normal \TeX category codes). Unless specifically required, expansion should be carried out using an appropriate argument specifier variant or the appropriate \exp_arg:N function.

\TeXhackers note: This is the \TeX primitive \expandafter renamed.
8 Preventing expansion

Despite the fact that the following functions are all about preventing expansion, they’re designed to be used in an expandable context and hence are all marked as being ‘expandable’ since they themselves will not appear after the expansion has completed.

\exp_not:N \exp_not:N \langle \text{token} \rangle

Prevents expansion of the \langle \text{token} \rangle in a context where it would otherwise be expanded, for example an x-type argument.

\TeXhackers note: This is the \TeX \texttt{noexpand} primitive.

\exp_not:c \exp_not:c \{\langle \text{tokens} \rangle\}

Expands the \langle \text{tokens} \rangle until only unexpandable content remains, and then converts this into a control sequence. Further expansion of this control sequence is then inhibited.

\exp_not:n \exp_not:n \{\langle \text{tokens} \rangle\}

Prevents expansion of the \langle \text{tokens} \rangle in a context where they would otherwise be expanded, for example an x-type argument.

\TeXhackers note: This is the \TeX \texttt{unexpanded} primitive. Hence its argument must be surrounded by braces.

\exp_not:V \exp_not:V \langle \text{variable} \rangle

Recovers the content of the \langle \text{variable} \rangle, then prevents expansion of this material in a context where it would otherwise be expanded, for example an x-type argument.

\exp_not:v \exp_not:v \{\langle \text{tokens} \rangle\}

Expands the \langle \text{tokens} \rangle until only unexpandable content remains, and then converts this into a control sequence (which should be a \langle \text{variable} \rangle name). The content of the \langle \text{variable} \rangle is recovered, and further expansion is prevented in a context where it would otherwise be expanded, for example an x-type argument.

\exp_not:o \exp_not:o \{\langle \text{tokens} \rangle\}

Expands the \langle \text{tokens} \rangle once, then prevents any further expansion in a context where they would otherwise be expanded, for example an x-type argument.

\exp_not:f \exp_not:f \{\langle \text{tokens} \rangle\}

Expands \langle \text{tokens} \rangle fully until the first unexpandable token is found. Expansion then stops, and the result of the expansion (including any tokens which were not expanded) is protected from further expansion.
This function terminates an \texttt{f}-type expansion. Thus if a function \texttt{f} starts an \texttt{f}-type expansion and all of \texttt{tokens} are expandable \texttt{\exp_stop_f:} will terminate the expansion of tokens even if \texttt{more tokens} are also expandable. The function itself is an implicit space token. Inside an \texttt{x}-type expansion, it will retain its form, but when typeset it produces the underlying space \texttt{(\_)}.

9 Internal functions and variables

\texttt{\texttt{\l__exp_internal_tl}}

The \texttt{\exp} module has its private variables to temporarily store results of the argument expansion. This is done to avoid interference with other functions using temporary variables.

\begin{verbatim}
\::n \cs_set_nopar:Npn \exp_args:Ncof { \::c \::o \::f \::: }
\::N \::p
\::c \::o \::f \::x \::v \::V \:::
\end{verbatim}

\texttt{\texttt{\cs_set_nopar:Npn \exp_args:Ncof { \::c \::o \::f \::: }}}

Internal forms for the base expansion types. These names do not conform to the general \texttt{\LaTeX} approach as this makes them more readily visible in the log and so forth.
Part VI
The l3prg package
Control structures

Conditional processing in \texttt{\LaTeX}3 is defined as something that performs a series of tests, possibly involving assignments and calling other functions that do not read further ahead in the input stream. After processing the input, a \textit{state} is returned. The typical states returned are \texttt{⟨true⟩} and \texttt{⟨false⟩} but other states are possible, say an \texttt{⟨error⟩} state for erroneous input, \textit{e.g.}, text as input in a function comparing integers.

\LaTeX{}3 has two forms of conditional flow processing based on these states. The first form is predicate functions that turn the returned state into a boolean \texttt{⟨true⟩} or \texttt{⟨false⟩}. For example, the function \texttt{\textbackslash if\_predicate:w} checks whether the control sequence given as its argument is free and then returns the boolean \texttt{⟨true⟩} or \texttt{⟨false⟩} values to be used in testing with \texttt{\if\_predicate:w} or in functions to be described below. The second form is the kind of functions choosing a particular argument from the input stream based on the result of the testing as in \texttt{\cs\_if\_free:NTF} which also takes one argument (the N) and then executes either \texttt{true} or \texttt{false} depending on the result. Important to note here is that the arguments are executed after exiting the underlying \texttt{\if...\fi:} structure.

1 Defining a set of conditional functions

\begin{verbatim}
\prg_new_conditional:Npnn \prg_new_conditional:Nnn \prg_set_conditional:Npnn \prg_set_conditional:Nnn
\prg_new_protected_conditional:Npnn \prg_new_protected_conditional:Nnn \prg_set_protected_conditional:Npnn \prg_set_protected_conditional:Nnn
\end{verbatim}

These functions create a family of conditionals using the same \texttt{⟨code⟩} to perform the test created. Those conditionals are expandable if \texttt{⟨code⟩} is. The \texttt{new} versions will check for existing definitions and perform assignments globally (\textit{cf.} \texttt{\cs\_new:Npn}) whereas the \texttt{set} versions do no check and perform assignments locally (\textit{cf.} \texttt{\cs\_set:Npn}). The conditionals created are dependent on the comma-separated list of \texttt{⟨conditions⟩}, which should be one or more of \texttt{p, T, F} and \texttt{TF}.

\begin{verbatim}
\prg_new_protected_conditional:Npnn \prg_new_protected_conditional:Nnn \prg_set_protected_conditional:Npnn \prg_set_protected_conditional:Nnn
\end{verbatim}

These functions create a family of protected conditionals using the same \texttt{⟨code⟩} to perform the test created. The \texttt{⟨code⟩} does not need to be expandable. The \texttt{new} version will check for existing definitions and perform assignments globally (\textit{cf.} \texttt{\cs\_new:Npn}) whereas the \texttt{set} version will not (\textit{cf.} \texttt{\cs\_set:Npn}). The conditionals created are dependent on the comma-separated list of \texttt{⟨conditions⟩}, which should be one or more of \texttt{T, F} and \texttt{TF} (not \texttt{p}).

36
The conditionals are defined by \texttt{\prg_new_conditional:Npnn} and friends as:

- \texttt{\langle name \rangle\_p:\langle arg spec \rangle} — a predicate function which will supply either a logical \texttt{true} or logical \texttt{false}. This function is intended for use in cases where one or more logical tests are combined to lead to a final outcome. This function will not work properly for protected conditionals.

- \texttt{\langle name \rangle:\langle arg spec \rangle T} — a function with one more argument than the original \texttt{\langle arg spec \rangle} demands. The \texttt{\langle true branch \rangle} code in this additional argument will be left on the input stream only if the test is \texttt{true}.

- \texttt{\langle name \rangle:\langle arg spec \rangle F} — a function with one more argument than the original \texttt{\langle arg spec \rangle} demands. The \texttt{\langle false branch \rangle} code in this additional argument will be left on the input stream only if the test is \texttt{false}.

- \texttt{\langle name \rangle:\langle arg spec \rangle TF} — a function with two more argument than the original \texttt{\langle arg spec \rangle} demands. The \texttt{\langle true branch \rangle} code in the first additional argument will be left on the input stream if the test is \texttt{true}, while the \texttt{\langle false branch \rangle} code in the second argument will be left on the input stream if the test is \texttt{false}.

The \texttt{\langle code \rangle} of the test may use \texttt{\langle parameters \rangle} as specified by the second argument to \texttt{\prg_set_conditional:Npnn}: this should match the \texttt{\langle argument specification \rangle} but this is not enforced. The \texttt{Nnn} versions infer the number of arguments from the argument specification given (cf. \texttt{\cs_new:Nn}, etc.). Within the \texttt{\langle code \rangle}, the functions \texttt{\prg_return_true:} and \texttt{\prg_return_false:} are used to indicate the logical outcomes of the test.

An example can easily clarify matters here:

\begin{verbatim}
\prg_set_conditional:Npnn \foo_if_bar:NN #1#2 { p , T , TF }
{ 
    \if_meaning:w \l_tmpa_tl #1 
    \prg_return_true: 
    \else: 
    \if_meaning:w \l_tmpa_tl #2 
    \prg_return_true: 
    \else: 
    \prg_return_false: 
    \fi: 
    \fi: 
}
\end{verbatim}

This defines the function \texttt{\foo_if_bar_p:NN}, \texttt{\foo_if_bar:NNTF} and \texttt{\foo_if_bar:NNT} but not \texttt{\foo_if_bar:NNF} (because \texttt{F} is missing from the \texttt{\langle conditions \rangle} list). The return statements take care of resolving the remaining \texttt{\else:} and \texttt{\fi:} before returning the state. There must be a return statement for each branch; failing to do so will result in erroneous output if that branch is executed.
\texttt{\textbackslash prg\_new\_eq\_conditional:NNn} \hspace{1em} \texttt{\textbackslash prg\_set\_eq\_conditional:NNn} \{\langle \text{conditions}\rangle\}

These functions copies a family of conditionals. The \texttt{new} version will check for existing definitions (\textit{cf.} \texttt{\textbackslash cs\_new:Npn}) whereas the \texttt{set} version will not (\textit{cf.} \texttt{\textbackslash cs\_set:Npn}). The conditionals copied are depended on the comma-separated list of \langle \text{conditions}\rangle, which should be one or more of \texttt{p, T, F} and \texttt{TF}.

\begin{itemize}
  \item \texttt{\textbackslash prg\_return\_true:}
  \item \texttt{\textbackslash prg\_return\_false:}
\end{itemize}

These ‘return’ functions define the logical state of a conditional statement. They appear within the code for a conditional function generated by \texttt{\textbackslash prg\_set\_conditional:Npnn}, \texttt{etc}, to indicate when a true or false branch has been taken. While they may appear multiple times each within the code of such conditionals, the execution of the conditional must result in the expansion of one of these two functions \textit{exactly once}.

The return functions trigger what is internally an \texttt{f}-expansion process to complete the evaluation of the conditional. Therefore, after \texttt{\textbackslash prg\_return\_true:} or \texttt{\textbackslash prg\_return\_false:} there must be no non-expandable material in the input stream for the remainder of the expansion of the conditional code. This includes other instances of either of these functions.

## 2 The boolean data type

This section describes a boolean data type which is closely connected to conditional processing as sometimes you want to execute some code depending on the value of a switch (\textit{e.g.}, draft/final) and other times you perhaps want to use it as a predicate function in an \texttt{\if\_predicate:w} test. The problem of the primitive \texttt{\if\_false:} and \texttt{\if\_true:} tokens is that it is not always safe to pass them around as they may interfere with scanning for termination of primitive conditional processing. Therefore, we employ two canonical booleans: \texttt{\c\_true\_bool} or \texttt{\c\_false\_bool}. Besides preventing problems as described above, it also allows us to implement a simple boolean parser supporting the logical operations And, Or, Not, \textit{etc.} which can then be used on both the boolean type and predicate functions.

All conditional \texttt{\bool\_} functions except assignments are expandable and expect the input to also be fully expandable (which will generally mean being constructed from predicate functions, possibly nested).

\begin{itemize}
  \item \texttt{\bool\_new:N} \langle \text{boolean}\rangle
  \item \texttt{\bool\_new:c}
\end{itemize}

Creates a new \langle \text{boolean}\rangle or raises an error if the name is already taken. The declaration is global. The \langle \text{boolean}\rangle will initially be \texttt{false}.

\begin{itemize}
  \item \texttt{\bool\_set\_false:N} \langle \text{boolean}\rangle
  \item \texttt{\bool\_set\_false:c}
  \item \texttt{\bool\_gset\_false:N}
  \item \texttt{\bool\_gset\_false:c}
\end{itemize}

Sets \langle \text{boolean}\rangle logically \texttt{false}.

38
\bool_set_true:N \bool_set_true:c \bool_gset_true:N \bool_gset_true:c

\bool_set_eq:NN \langle boolean \rangle \langle boolean \rangle

Sets the content of \langle boolean_1 \rangle equal to that of \langle boolean_2 \rangle.

\bool_set:Nn \langle boolean \rangle \{ \langle boolean expression \rangle \}

Evaluates the \langle boolean expression \rangle as described for \bool_if:n(TF), and sets the \langle boolean \rangle variable to the logical truth of this evaluation.

\bool_if_p:N \bool_if_p:c \bool_if:NTF \bool_if:c

Tests the current truth of \langle boolean \rangle, and continues expansion based on this result.

\bool_show:N \bool_show:c

Displays the logical truth of the \langle boolean \rangle on the terminal.

\bool_show:n

Displays the logical truth of the \langle boolean expression \rangle on the terminal.

\bool_if_exist_p:N \bool_if_exist:NTF \bool_if_exist:c

Tests whether the \langle boolean \rangle is currently defined. This does not check that the \langle boolean \rangle really is a boolean variable.

\l_tmpa_bool \l_tmpb_bool

A scratch boolean for local assignment. It is never used by the kernel code, and so is safe for use with any \LaTeX-defined function. However, it may be overwritten by other non-kernel code and so should only be used for short-term storage.

\g_tmpa_bool \g_tmpb_bool

A scratch boolean for global assignment. It is never used by the kernel code, and so is safe for use with any \LaTeX-defined function. However, it may be overwritten by other non-kernel code and so should only be used for short-term storage.
3 Boolean expressions

As we have a boolean datatype and predicate functions returning boolean \langle true \rangle or \langle false \rangle values, it seems only fitting that we also provide a parser for \langle boolean expressions \rangle.

A boolean expression is an expression which given input in the form of predicate functions and boolean variables, return boolean \langle true \rangle or \langle false \rangle. It supports the logical operations And, Or and Not as the well-known infix operators &&, || and ! with their usual precedences. In addition to this, parentheses can be used to isolate sub-expressions. For example,

\begin{verbatim}
\int_compare_p:n \{ 1 = 1 \} &&
 ( 
 \int_compare_p:n \{ 2 = 3 \} ||
 \int_compare_p:n \{ 4 = 4 \} ||
 \int_compare_p:n \{ 1 = \error \} % is skipped
 ) &&
! ( \int_compare_p:n \{ 2 = 4 \} )
\end{verbatim}

is a valid boolean expression. Note that minimal evaluation is carried out whenever possible so that whenever a truth value cannot be changed any more, the remaining tests within the current group are skipped.

\begin{verbatim}
\bool_if_p:n \{ (boolean expression) \}
\bool_if:nTF \{ (boolean expression) \} \{ (true code) \} \{ (false code) \}
\end{verbatim}

Tests the current truth of \langle boolean expression \rangle, and continues expansion based on this result. The \langle boolean expression \rangle should consist of a series of predicates or boolean variables with the logical relationship between these defined using && (“And”), || (“Or”), ! (“Not”) and parentheses. Minimal evaluation is used in the processing, so that once a result is defined there is not further expansion of the tests. For example

\begin{verbatim}
\bool_if_p:n
{ 
 \int_compare_p:nNn \{ 1 \} = \{ 1 \}
 &&
 ( 
 \int_compare_p:nNn \{ 2 \} = \{ 3 \} ||
 \int_compare_p:nNn \{ 4 \} = \{ 4 \} ||
 \int_compare_p:nNn \{ 1 \} = \{ \error \} % is skipped
 )
 &&
! \int_compare_p:nNn \{ 2 \} = \{ 4 \}
}
\end{verbatim}

will be true and will not evaluate \int_compare_p:nNn \{ 1 \} = \{ \error \}. The logical Not applies to the next predicate or group.
4 Logical loops

Loops using either boolean expressions or stored boolean values.

\bool_do_until:Nn \bool_do_until:Nn \bool_do_until:cn
\bool_do_until:Nn \bool_do_until:cn
Places the \langle code\rangle in the input stream for \texttt{TeX} to process, and then checks the logical value of the \langle boolean\rangle. If it is \texttt{false} then the \langle code\rangle will be inserted into the input stream again and the process will loop until the \langle boolean\rangle is \texttt{true}.

\bool_do_while:Nn \bool_do_while:Nn \bool_do_while:cn
\bool_do_while:Nn \bool_do_while:cn
Places the \langle code\rangle in the input stream for \texttt{TeX} to process, and then checks the logical value of the \langle boolean\rangle. If it is \texttt{true} then the \langle code\rangle will be inserted into the input stream again and the process will loop until the \langle boolean\rangle is \texttt{false}.

\bool_until_do:Nn \bool_until_do:Nn \bool_until_do:cn
\bool_until_do:Nn \bool_until_do:cn
This function firsts checks the logical value of the \langle boolean\rangle. If it is \texttt{false} the \langle code\rangle is placed in the input stream and expanded. After the completion of the \langle code\rangle the truth of the \langle boolean\rangle is re-evaluated. The process will then loop until the \langle boolean\rangle is \texttt{true}.

\bool_while_do:Nn \bool_while_do:Nn \bool_while_do:cn
\bool_while_do:Nn \bool_while_do:cn
This function firsts checks the logical value of the \langle boolean\rangle. If it is \texttt{true} the \langle code\rangle is placed in the input stream and expanded. After the completion of the \langle code\rangle the truth of the \langle boolean\rangle is re-evaluated. The process will then loop until the \langle boolean\rangle is \texttt{false}.

\bool_do_until:nn \bool_do_until:nn \bool_do_until:cn
\bool_do_until:nn \bool_do_until:cn
Places the \langle code\rangle in the input stream for \texttt{TeX} to process, and then checks the logical value of the \langle boolean expression\rangle as described for \texttt{\bool_if:nTF}. If it is \texttt{false} then the \langle code\rangle will be inserted into the input stream again and the process will loop until the \langle boolean expression\rangle evaluates to \texttt{true}.

\bool_do_while:nn \bool_do_while:nn \bool_do_while:cn
\bool_do_while:nn \bool_do_while:cn
Places the \langle code\rangle in the input stream for \texttt{TeX} to process, and then checks the logical value of the \langle boolean expression\rangle as described for \texttt{\bool_if:nTF}. If it is \texttt{true} then the \langle code\rangle will be inserted into the input stream again and the process will loop until the \langle boolean expression\rangle evaluates to \texttt{false}.
This function first checks the logical value of the \textit{boolean expression} (as described for \texttt{\textbackslash bool_if:nTF}). If it is false the \texttt{(code)} is placed in the input stream and expanded. After the completion of the \texttt{(code)} the truth of the \textit{boolean expression} is re-evaluated. The process will then loop until the \textit{boolean expression} is true.

This function first checks the logical value of the \textit{boolean expression} (as described for \texttt{\textbackslash bool_if:nTF}). If it is true the \texttt{(code)} is placed in the input stream and expanded. After the completion of the \texttt{(code)} the truth of the \textit{boolean expression} is re-evaluated. The process will then loop until the \textit{boolean expression} is false.

\section{Producing multiple copies}

\texttt{\textbackslash prg_replicate:nn} \{\textit{integer expression}\} \{\texttt{(tokens)}\}

Evaluates the \textit{integer expression} (which should be zero or positive) and creates the resulting number of copies of the \texttt{(tokens)}. The function is both expandable and safe for nesting. It yields its result after two expansion steps.

\section{Detecting \TeX{}'s mode}

\texttt{\textbackslash mode_if_horizontal_p:} \texttt{\textbackslash mode_if_horizontal:TF} \{\texttt{(true code)}\} \{\texttt{(false code)}\}

Detects if \TeX{} is currently in horizontal mode.

\texttt{\textbackslash mode_if_inner_p:} \texttt{\textbackslash mode_if_inner:TF} \{\texttt{(true code)}\} \{\texttt{(false code)}\}

Detects if \TeX{} is currently in inner mode.

\texttt{\textbackslash mode_if_math_p:} \texttt{\textbackslash mode_if_math:TF} \{\texttt{(true code)}\} \{\texttt{(false code)}\}

Detects if \TeX{} is currently in maths mode.

\texttt{\textbackslash mode_if_vertical_p:} \texttt{\textbackslash mode_if_vertical:TF} \{\texttt{(true code)}\} \{\texttt{(false code)}\}

Detects if \TeX{} is currently in vertical mode.
7 Primitive conditionals

\if_predicate:w \langle predicate \rangle \{ true code \} \else: \{ false code \} \fi:

This function takes a predicate function and branches according to the result. (In practice this function would also accept a single boolean variable in place of the \langle predicate \rangle but to make the coding clearer this should be done through \if_bool:N.)

\if_bool:N \langle boolean \rangle \{ true code \} \else: \{ false code \} \fi:

This function takes a boolean variable and branches according to the result.

8 Internal programming functions

\group_align_safe_begin:  
\group_align_safe_end:  

These functions are used to enclose material in a \TeX{} alignment environment within a specially-constructed group. This group is designed in such a way that it does not add brace groups to the output but does act as a group for the & token inside \halign. This is necessary to allow grabbing of tokens for testing purposes, as \TeX{} uses group level to determine the effect of alignment tokens. Without the special grouping, the use of a function such as \peek_after:Nw will result in a forbidden comparison of the internal \endtemplate token, yielding a fatal error. Each \group_align_safe_begin: must be matched by a \group_align_safe_end:, although this does not have to occur within the same function.

\scan_align_safe_stop:  

Stops \TeX{}’s scanner looking for expandable control sequences at the beginning of an alignment cell. This function is required, for example, to obtain the expected output when testing \mode_if_math:TF at the start of a math array cell: placing \scan_align_safe_stop: before \mode_if_math:TF will give the correct result. This function does not destroy any kerning if used in other locations, but does render functions non-expandable.

\TeX{}Xhackers note: This is a protected version of \prg_do_nothing:, which therefore stops \TeX{}’s scanner in the circumstances described without producing any affect on the output.

\__prg_variable_get_scope:N \langle variable \rangle  

Returns the scope (g for global, blank otherwise) for the \langle variable \rangle.

\__prg_variable_get_type:N \langle variable \rangle  

Returns the type of \langle variable \rangle (tl, int, etc.)
\__prg_break_point:Nn \__prg_break_point:Nn \(\text{type}\)\_map\_break: \(\text{tokens}\)

Used to mark the end of a recursion or mapping: the functions \(\text{type}\)\_map\_break: and \(\text{type}\)\_map\_break:n use this to break out of the loop. After the loop ends, the \(\text{tokens}\) are inserted into the input stream. This occurs even if the break functions are not applied: \__prg_break_point:Nn is functionally-equivalent in these cases to \use_ii:nn.

\__prg_map_break:Nn \__prg_map_break:Nn \(\text{type}\)\_map\_break: \{\text{user code}\}

\__prg_map_break:Nn \(\text{type}\)\_map\_break: \{\text{ending code}\}

Breaks a recursion in mapping contexts, inserting in the input stream the \(\text{user code}\) after the \(\text{ending code}\) for the loop. The function breaks loops, inserting their \(\text{ending code}\), until reaching a loop with the same \(\text{type}\) as its first argument. This \(\text{type}\)\_map\_break: argument is simply used as a recognizable marker for the \(\text{type}\).

\g__prg_map_int

This integer is used by non-expandable mapping functions to track the level of nesting in force. The functions \__prg_map_1:w, \__prg_map_2:w, etc., labelled by \g__prg_map_int hold functions to be mapped over various list datatypes in inline and variable mappings.

\__prg_break_point: \__prg_break_point:

This copy of \prg_do_nothing: is used to mark the end of a fast short-term recursions: the function \__prg_break:n uses this to break out of the loop.

\__prg_break: \__prg_break:n \{\text{tokens}\} \ldots \__prg_break_point:

Breaks a recursion which has no \(\text{ending code}\) and which is not a user-breakable mapping (see for instance \prop_get:Nn), and inserts \(\text{tokens}\) in the input stream.
Part VII
The \texttt{l3quark} package

Quarks

1 Introduction to quarks and scan marks

Two special types of constants in LaTeX are "quarks" and "scan marks". By convention all constants of type quark start out with $\texttt{\q_}$, and scan marks start with $\texttt{\s_}$. Scan marks are for internal use by the kernel: they are not intended for more general use.

1.1 Quarks

Quarks are control sequences that expand to themselves and should therefore never be executed directly in the code. This would result in an endless loop!

They are meant to be used as delimiter in weird functions, with the most command use case as the ‘stop token’ (\textit{i.e.} $\texttt{\q_stop}$). For example, when writing a macro to parse a user-defined date

\begin{verbatim}
\date_parse:n {19/June/1981}
\end{verbatim}

one might write a command such as

\begin{verbatim}
\cs_new:Npn \date_parse:n #1 \q_stop \cs_new:Npn \date_parse_aux:w #1 / #2 / #3 \q_stop
{ <do something with the date> }
\end{verbatim}

Quarks are sometimes also used as error return values for functions that receive erroneous input. For example, in the function $\texttt{\prop:get:NnN}$ to retrieve a value stored in some key of a property list, if the key does not exist then the return value is the quark $\texttt{\q_no_value}$. As mentioned above, such quarks are extremely fragile and it is imperative when using such functions that code is carefully written to check for pathological cases to avoid leakage of a quark into an uncontrolled environment.

Quarks also permit the following ingenious trick when parsing tokens: when you pick up a token in a temporary variable and you want to know whether you have picked up a particular quark, all you have to do is compare the temporary variable to the quark using $\texttt{\tl_if_eq:NNTF}$. A set of special quark testing functions is set up below. All the quark testing functions are expandable although the ones testing only single tokens are much faster. An example of the quark testing functions and their use in recursion can be seen in the implementation of $\texttt{\clist_map_function:NN}$.
2 Defining quarks

\quark_new:N \quark_new:N \langle \quark \rangle

Creates a new \langle \quark \rangle which expands only to \langle \quark \rangle. The \langle \quark \rangle will be defined globally, and an error message will be raised if the name was already taken.

\q_stop

Used as a marker for delimited arguments, such as

\cs_set:Npn \tmp:w #1#2 \q_stop \{#1\}

\q_mark

Used as a marker for delimited arguments when \q_stop is already in use.

\q_nil

Quark to mark a null value in structured variables or functions. Used as an end delimiter when this may itself may need to be tested (in contrast to \q_stop, which is only ever used as a delimiter).

\q_no_value

A canonical value for a missing value, when one is requested from a data structure. This is therefore used as a “return” value by functions such as \prop_get:NnN if there is no data to return.

3 Quark tests

The method used to define quarks means that the single token (N) tests are faster than the multi-token (n) tests. The later should therefore only be used when the argument can definitely take more than a single token.

\quark_if_nil_p:N \quark_if_nil_p:N \langle \text{token} \rangle\n\quark_if_nil_p:N \langle \text{token} \rangle \{ \langle \text{true code} \rangle \} \{ \langle \text{false code} \rangle \}

Tests if the \langle \text{token} \rangle is equal to \langle \text{q_nil} \rangle.

\quark_if_nil_p:n \quark_if_nil:nTF \langle \text{token list} \rangle\n\quark_if_nil_p:n \langle \text{token list} \rangle \{ \langle \text{true code} \rangle \} \{ \langle \text{false code} \rangle \}

Tests if the \langle \text{token list} \rangle contains only \langle \text{q_nil} \rangle (distinct from \langle \text{token list} \rangle being empty or containing \langle \text{q_nil} \rangle plus one or more other tokens).

\quark_if_no_value_p:N \quark_if_no_value_p:c \quark_if_no_value:N \quark_if_no_value:cTF
\quark_if_no_value_p:N \langle \text{token} \rangle\n\quark_if_no_value_p:n \langle \text{token list} \rangle\n\quark_if_no_value_p:n \langle \text{token list} \rangle \{ \langle \text{true code} \rangle \} \{ \langle \text{false code} \rangle \}

Tests if the \langle \text{token} \rangle is equal to \langle \text{q_no_value} \rangle.

\quark_if_no_value:nTF \langle \text{token list} \rangle\n\quark_if_no_value:nTF \langle \text{token list} \rangle \{ \langle \text{true code} \rangle \} \{ \langle \text{false code} \rangle \}

Tests if the \langle \text{token list} \rangle contains only \langle \text{q_no_value} \rangle (distinct from \langle \text{token list} \rangle being empty or containing \langle \text{q_no_value} \rangle plus one or more other tokens).
4 Recursion

This module provides a uniform interface to intercepting and terminating loops as when one is doing tail recursion. The building blocks follow below and an example is shown in Section 5.

\q_recursion_tail

This quark is appended to the data structure in question and appears as a real element there. This means it gets any list separators around it.

\q_recursion_stop

This quark is added after the data structure. Its purpose is to make it possible to terminate the recursion at any point easily.

\quark_if_recursion_tail_stop:N  \quark_if_recursion_tail_stop:N \langle token \rangle

Tests if \langle token \rangle contains only the marker \q_recursion_tail, and if so terminates the recursion this is part of using \use_none_delimit_by_q_recursion_stop:w. The recursion input must include the marker tokens \q_recursion_tail and \q_recursion_stop as the last two items.

\quark_if_recursion_tail_stop:n \langle token list \rangle

\quark_if_recursion_tail_stop:o

Tests if the \langle token list \rangle contains only \q_recursion_tail, and if so terminates the recursion this is part of using \use_none_delimit_by_q_recursion_stop:w. The recursion input must include the marker tokens \q_recursion_tail and \q_recursion_stop as the last two items.

\quark_if_recursion_tail_stop_do:Nn \langle insertion \rangle

Tests if \langle token \rangle contains only \q_recursion_tail, and if so terminates the recursion this is part of using \use_none_delimit_by_q_recursion_stop:w. The recursion input must include the marker tokens \q_recursion_tail and \q_recursion_stop as the last two items. The \langle insertion \rangle code is then added to the input stream after the recursion has ended.

\quark_if_recursion_tail_stop_do:nn \langle token list \rangle \langle insertion \rangle

\quark_if_recursion_tail_stop_do:on

Tests if the \langle token list \rangle contains only \q_recursion_tail, and if so terminates the recursion this is part of using \use_none_delimit_by_q_recursion_stop:w. The recursion input must include the marker tokens \q_recursion_tail and \q_recursion_stop as the last two items. The \langle insertion \rangle code is then added to the input stream after the recursion has ended.
5 An example of recursion with quarks

Quarks are mainly used internally in the expl3 code to define recursion functions such as \texttt{tl_map_inline:nn} and so on. Here is a small example to demonstrate how to use quarks in this fashion. We shall define a command called \texttt{my_map_dbl:nn} which takes a token list and applies an operation to every pair of tokens. For example, \texttt{my_map_dbl:nn \{abcd\} \{[--#1--#2--]\}--} would produce \texttt{[\textendash a\textendash b--] [\textendash c\textendash d--] }. Using quarks to define such functions simplifies their logic and ensures robustness in many cases.

Here’s the definition of \texttt{my_map_dbl:nn}. First of all, define the function that will do the processing based on the inline function argument \texttt{#2}. Then initiate the recursion using an internal function. The token list \texttt{#1} is terminated using \texttt{q_recursion_tail}, with delimiters according to the type of recursion (here a pair of \texttt{q_recursion_tail}), concluding with \texttt{q_recursion_stop}. These quarks are used to mark the end of the token list being operated upon.

\begin{verbatim}
\cs_new:Npn \my_map_dbl:nn #1#2
{\cs_set:Npn \__my_map_dbl_fn:nn ##1 ##2 {#2}\__my_map_dbl:nn #1 \q_recursion_tail \q_recursion_tail \q_recursion_stop}
\end{verbatim}

The definition of the internal recursion function follows. First check if either of the input tokens are the termination quarks. Then, if not, apply the inline function to the two arguments.

\begin{verbatim}
\cs_new:Nn \__my_map_dbl:nn
{\quark_if_recursion_tail_stop:nN {#1}\quark_if_recursion_tail_stop:nN \__my_map_dbl_fn:nn \#1 \#2 \q_recursion_tail \q_recursion_stop}\__my_map_dbl:nn}
\end{verbatim}

Finally, recurse:

\begin{verbatim}
\__my_map_dbl:nn}
\end{verbatim}

Note that contrarily to \LaTeX3 built-in mapping functions, this mapping function cannot be nested, since the second map will overwrite the definition of \texttt{\_\_my_map_dbl_fn:nn}.

6 Internal quark functions

\begin{verbatim}
\_\_quark_if_recursion_tail_break:NN \_\_quark_if_recursion_tail_break:nN \{token list\}\_\_quark_if_recursion_tail_break:NN \{type\}_map_break:
\_\_quark_if_recursion_tail_break:NN \{type\}_map_break:.
\end{verbatim}

Tests if \texttt{(token list)} contains only \texttt{q_recursion_tail}, and if so terminates the recursion using \texttt{(type)}\_map_break:. The recursion end should be marked by \texttt{prg_break_point:NN (type)}\_map_break:.
7 Scan marks

Scan marks are control sequences set equal to `\scan_stop:`, hence will never expand in an expansion context and will be (largely) invisible if they are encountered in a typesetting context.

Like quarks, they can be used as delimiters in weird functions and are often safer to use for this purpose. Since they are harmless when executed by \TeX in non-expandable contexts, they can be used to mark the end of a set of instructions. This allows to skip to that point if the end of the instructions should not be performed (see \texttt{l3regex}).

The scan marks system is only for internal use by the kernel team in a small number of very specific places. These functions should not be used more generally.

\begin{verbatim}
\__scan_new:N \__scan_new:N \scan_stop: \scan_stop:
\end{verbatim}

Creates a new \texttt{(scan mark)} which is set equal to `\scan_stop:`. The \texttt{(scan mark)} will be defined globally, and an error message will be raised if the name was already taken by another scan mark.

\begin{verbatim}
\s__stop \__use_none_delimit_by_s__stop:w
\end{verbatim}

Used at the end of a set of instructions, as a marker that can be jumped to using \texttt{\__use_none_delimit_by_s__stop:w}.

\begin{verbatim}
\__use_none_delimit_by_s__stop:w \__use_none_delimit_by_s__stop:w \tokens \s__stop
\end{verbatim}

Removes the \texttt{(tokens)} and \texttt{\s__stop} from the input stream. This leads to a low-level \TeX error if \texttt{\s__stop} is absent.
Part VIII
The l3token package
Token manipulation

This module deals with tokens. Now this is perhaps not the most precise description so let's try with a better description: When programming in \TeX, it is often desirable to know just what a certain token is: is it a control sequence or something else. Similarly one often needs to know if a control sequence is expandable or not, a macro or a primitive, how many arguments it takes etc. Another thing of great importance (especially when it comes to document commands) is looking ahead in the token stream to see if a certain character is present and maybe even remove it or disregard other tokens while scanning. This module provides functions for both and as such will have two primary function categories: \token_ for anything that deals with tokens and \peek_ for looking ahead in the token stream.

Most of the time we will be using the term “token” but most of the time the function we're describing can equally well be used on a control sequence as such one is one token as well.

We shall refer to list of tokens as tlists and such lists represented by a single control sequence is a “token list variable” tl var. Functions for these two types are found in the l3tl module.

1 All possible tokens

Let us start by reviewing every case that a given token can fall into. It is very important to distinguish two aspects of a token: its meaning, and what it looks like.

For instance, \if:w, \if_charcode:w, and \tex_if:D are three for the same internal operation of \TeX, namely the primitive testing the next two characters for equality of their character code. They behave identically in many situations. However, \TeX distinguishes them when searching for a delimited argument. Namely, the example function \show_until_if:w defined below will take everything until \if:w as an argument, despite the presence of other copies of \if:w under different names.

\cs_new:Npn \show_until_if:w #1 \if:w { \tl_show:n {#1} }
\show_until_if:w \tex_if:D \if_charcode:w \if:w

\show_until_if:w #1 \if:w { \tl_show:n {#1} }
\show_until_if:w \tex_if:D \if_charcode:w \if:w
2 Character tokens

\char_set_catcode_escape:N \char_set_catcode_letter:N \langle character \rangle
\char_set_catcode_group_begin:N \char_set_catcode_group_end:N
\char_set_catcode_math-toggle:N \char_set_catcode_alignment:N
\char_set_catcode_end_line:N \char_set_catcode_parameter:N
\char_set_catcode_math-superscript:N \char_set_catcode_math-subscript:N
\char_set_catcode_ignore:N \char_set_catcode_space:N
\char_set_catcode_letter:N \char_set_catcode_other:N
\char_set_catcode_active:N \char_set_catcode_comment:N
\char_set_catcode_invalid:N

Sets the category code of the \langle character \rangle to that indicated in the function name. Depending on the current category code of the \langle token \rangle the escape token may also be needed:

\char_set_catcode_other:N \%

The assignment is local.

\char_set_catcode_escape:n \char_set_catcode_letter:n \langle \textit{integer expression} \rangle
\char_set_catcode_group_begin:n \char_set_catcode_group_end:n
\char_set_catcode_math-toggle:n \char_set_catcode_alignment:n
\char_set_catcode_end_line:n \char_set_catcode_parameter:n
\char_set_catcode_math-superscript:n \char_set_catcode_math-subscript:n
\char_set_catcode_ignore:n \char_set_catcode_space:n
\char_set_catcode_letter:n \char_set_catcode_other:n
\char_set_catcode_active:n \char_set_catcode_comment:n
\char_set_catcode_invalid:n

Sets the category code of the \langle character \rangle which has character code as given by the \langle \textit{integer expression} \rangle. This version can be used to set up characters which cannot otherwise be given (cf. the \texttt{N}-type variants). The assignment is local.
These functions set the category code of the \textit{character} which has character code as
given by the \textit{integer expression}. The first \textit{integer expression} is the character code
and the second is the category code to apply. The setting applies within the current \TeX{} group. In general, the symbolic functions \texttt{\char_set_catcode\_\{type\}} should be
preferred, but there are cases where these lower-level functions may be useful.

\texttt{\char_value_catcode:n} \texttt{\char_value_catcode:n \{\textit{integer expression}\}}

Expands to the current category code of the \textit{character} with character code given by
the \textit{integer expression}.

\texttt{\char_show_value_catcode:n}

Displays the current category code of the \textit{character} with character code given by the
\textit{integer expression} on the terminal.

\texttt{\char_set_lccode:nn} \texttt{\char_set_lccode:nn \{\textit{expr}_1\} \{\textit{expr}_2\}}

Sets up the behaviour of the \textit{character} when found inside \texttt{\tl_to_lowercase:n}, such
that \textit{character}_1 will be converted into \textit{character}_2. The two \textit{characters} may be spec-
fified using an \textit{integer expression} for the character code concerned. This may include
the \TeX{} \texttt{\textit{character}} method for converting a single character into its character code:

\begin{verbatim}
\char_set_lccode:nn \{‘\texttt{\textbackslash A}\} \{‘\texttt{\textbackslash a}\} % Standard behaviour
\char_set_lccode:nn \{‘\texttt{\textbackslash A}\} \{‘\texttt{\textbackslash A + 32}\}
\char_set_lccode:nn \{50\} \{60\}
\end{verbatim}

The setting applies within the current \TeX{} group.

\texttt{\char_value_lccode:n} \texttt{\char_value_lccode:n \{\textit{integer expression}\}}

Expands to the current lower case code of the \textit{character} with character code given by
the \textit{integer expression}.

\texttt{\char_show_value_lccode:n}

Displays the current lower case code of the \textit{character} with character code given by the
\textit{integer expression} on the terminal.
\char_set_uccode:nn \char_set_uccode:nn {{\texttt{intexpr}_1}} {{\texttt{intexpr}_2}}

Sets up the behaviour of the \texttt{character} when found inside \texttt{tl_to_uppercase:n}, such that \texttt{character}_1 will be converted into \texttt{character}_2. The two \texttt{characters} may be specified using an \texttt{integer expression} for the character code concerned. This may include the \TeX\ '\texttt{character}' method for converting a single character into its character code:

\begin{verbatim}
\char_set_uccode:nn { '\a } { '\A } % Standard behaviour
\char_set_uccode:nn { '\A } { '\A - 32 }
\char_set_uccode:nn { 60 } { 50 }
\end{verbatim}

The setting applies within the current \TeX\ group.

\char_value_uccode:n \char_value_uccode:n {{\texttt{integer expression}}}

Expands to the current upper case code of the \texttt{character} with character code given by the \texttt{integer expression}.

\char_show_value_uccode:n \char_show_value_uccode:n {{\texttt{integer expression}}}

Displays the current upper case code of the \texttt{character} with character code given by the \texttt{integer expression} on the terminal.

\char_set_mathcode:nn \char_set_mathcode:nn {{\texttt{intexpr}_1}} {{\texttt{intexpr}_2}}

This function sets up the math code of \texttt{character}. The \texttt{character} is specified as an \texttt{integer expression} which will be used as the character code of the relevant character. The setting applies within the current \TeX\ group.

\char_value_mathcode:n \char_value_mathcode:n {{\texttt{integer expression}}}

Expands to the current math code of the \texttt{character} with character code given by the \texttt{integer expression}.

\char_show_value_mathcode:n \char_show_value_mathcode:n {{\texttt{integer expression}}}

Displays the current math code of the \texttt{character} with character code given by the \texttt{integer expression} on the terminal.

\char_set_sfcode:nn \char_set_sfcode:nn {{\texttt{intexpr}_1}} {{\texttt{intexpr}_2}}

This function sets up the space factor for the \texttt{character}. The \texttt{character} is specified as an \texttt{integer expression} which will be used as the character code of the relevant character. The setting applies within the current \TeX\ group.

\char_value_sfcode:n \char_value_sfcode:n {{\texttt{integer expression}}}

Expands to the current space factor for the \texttt{character} with character code given by the \texttt{integer expression}.
\char_show_value sfcode:n \char_show_value sfcode:n \{\texttt{integer expression}\}

Displays the current space factor for the \texttt{character} with character code given by the \texttt{integer expression} on the terminal.

\l_char_active_seq

New: 2012-01-23

Used to track which tokens will require special handling at the document level as they are of category \texttt{active} (catcode 13). Each entry in the sequence consists of a single active character. Active tokens should be added to the sequence when they are defined for general document use.

\l_char_special_seq

New: 2012-01-23

Used to track which tokens will require special handling when working with verbatim-like material at the document level as they are not of categories \texttt{letter} (catcode 11) or \texttt{other} (catcode 12). Each entry in the sequence consists of a single escaped token, for example \texttt{\} for the backslash or \texttt{\{ for an opening brace.} Escaped tokens should be added to the sequence when they are defined for general document use.

3 Generic tokens

\token_new:Nn \token_new:Nn \{\texttt{token}_1\} \{\texttt{token}_2\}

Defines \texttt{token}_1 to globally be a snapshot of \texttt{token}_2. This will be an implicit representation of \texttt{token}_2.

These are implicit tokens which have the category code described by their name. They are used internally for test purposes but are also available to the programmer for other uses.

\c_group_begin_token \c_group_end_token \c_math_toggle_token \c_alignment_token \c_parameter_token \c_math_superscript_token \c_math_subscript_token \c_space_token

These are implicit tokens which have the category code described by their name. They are used internally for test purposes and should not be used other than for category code tests.

\c_catcode_letter_token \c_catcode_other_token

A token list containing an active token. This is used internally for test purposes and should not be used other than in appropriately-constructed category code tests.
4 Converting tokens

\token_to_meaning:N \token_to_meaning:c
Inserts the current meaning of the \token into the input stream as a series of characters of category code 12 (other). This will be the primitive \TeX description of the \token, thus for example both functions defined by \cs_set_nopar:Npn and token list variables defined using \tl_new:N will be described as macros.

\TeXhackers note: This is the \TeX primitive \meaning.

\token_to_str:N \token_to_str:c
Converts the given \token into a series of characters with category code 12 (other). The current escape character will be the first character in the sequence, although this will also have category code 12 (the escape character is part of the \token). This function requires only a single expansion.

\TeXhackers note: \token_to_str:N is the \TeX primitive \string renamed.

5 Token conditionals

\token_if_group_begin_p:N \token_if_group_begin_p:N \token_if_group_end_p:N \token_if_group_end_p:N \token_if_math_toggle_p:N \token_if_math_toggle_p:N \token_if_alignment_p:N \token_if_alignment_p:N
Tests if \token has the category code of a begin group token (\{ when normal \TeX category codes are in force). Note that an explicit begin group token cannot be tested in this way, as it is not a valid \N-type argument.

Tests if \token has the category code of an end group token (\} when normal \TeX category codes are in force). Note that an explicit end group token cannot be tested in this way, as it is not a valid \N-type argument.

Tests if \token has the category code of a math shift token ($ when normal \TeX category codes are in force).

Tests if \token has the category code of an alignment token (& when normal \TeX category codes are in force).
\token_if_parameter_p:N \token_if_parameter_p:N \token_if_alignment:NTF \token_if_parameter:N \token_if_math_superscript_p:N \token_if_math_superscript:NTF \token_if_math_subscript_p:N \token_if_math_subscript:NTF \token_if_space_p:N \token_if_space:NTF \token_if_letter_p:N \token_if_letter:NTF \token_if_other_p:N \token_if_other:NTF \token_if_active_p:N \token_if_active:NTF \token_if_eq_catcode_p:NN \token_if_eq_catcode:NNTF \token_if_eq_charcode_p:NN \token_if_eq_charcode:NNTF

Tests if \token has the category code of a macro parameter token (# when normal \TeX\ category codes are in force).

Tests if \token has the category code of a superscript token (^ when normal \TeX\ category codes are in force).

Tests if \token has the category code of a subscript token (\_ when normal \TeX\ category codes are in force).

Tests if \token has the category code of a space token. Note that an explicit space token with character code 32 cannot be tested in this way, as it is not a valid N-type argument.

Tests if \token has the category code of a letter token.

Tests if \token has the category code of an “other” token.

Tests if \token has the category code of an active character.

Tests if the two \tokens have the same category code.

Tests if the two \tokens have the same character code.
Tests if the two \emph{tokens} have the same meaning when expanded.

Tests if the \emph{token} is a \TeX\ macro.

Tests if the \emph{token} is a control sequence.

Tests if the \emph{token} is expandable. This test returns \emph{false} for an undefined token.

Tests if the \emph{token} is a long macro.

Tests if the \emph{token} is a protected macro: a macro which is both protected and long will return logical \emph{false}.

Tests if the \emph{token} is a protected long macro.

Tests if the \emph{token} is defined to be a chardef.

\TeXhackers note: Booleans, boxes and small integer constants are implemented as chardefs.
Tests if the \langle token \rangle is defined to be a mathchardef.

Tests if the \langle token \rangle is defined to be a dimension register.

Tests if the \langle token \rangle is defined to be a integer register.

**\TeXhackers note:** Constant integers may be implemented as integer registers, chardefs, or mathchardefs depending on their value.

Tests if the \langle token \rangle is defined to be a muskip register.

Tests if the \langle token \rangle is defined to be a skip register.

Tests if the \langle token \rangle is defined to be a toks register (not used by \TeX3).

Tests if the \langle token \rangle is an engine primitive.
6 Peeking ahead at the next token

There is often a need to look ahead at the next token in the input stream while leaving it in place. This is handled using the “peek” functions. The generic \texttt{\textbackslash peek\_after:Nw} is provided along with a family of predefined tests for common cases. As peeking ahead does not skip spaces the predefined tests include both a space-respecting and space-skipping version.

\eqn{\texttt{\textbackslash peek\_after:Nw}}<\texttt{\textbackslash token}\rangle \langle\texttt{\textbackslash function}\rangle

Locally sets the test variable \texttt{\textbackslash l\_peek\_token} equal to \langle\texttt{\textbackslash token}\rangle (as an implicit token, not as a token list), and then expands the \langle\texttt{\textbackslash function}\rangle. The \langle\texttt{\textbackslash token}\rangle will remain in the input stream as the next item after the \langle\texttt{\textbackslash function}\rangle. The \langle\texttt{\textbackslash token}\rangle here may be \texttt{\textbackslash }, \{ or \} (assuming normal \TeX category codes), i.e. it is not necessarily the next argument which would be grabbed by a normal function.

\eqn{\texttt{\textbackslash peek\_gafter:Nw}}<\texttt{\textbackslash token}\rangle \langle\texttt{\textbackslash function}\rangle

Globally sets the test variable \texttt{\textbackslash g\_peek\_token} equal to \langle\texttt{\textbackslash token}\rangle (as an implicit token, not as a token list), and then expands the \langle\texttt{\textbackslash function}\rangle. The \langle\texttt{\textbackslash token}\rangle will remain in the input stream as the next item after the \langle\texttt{\textbackslash function}\rangle. The \langle\texttt{\textbackslash token}\rangle here may be \texttt{\textbackslash }, \{ or \} (assuming normal \TeX category codes), i.e. it is not necessarily the next argument which would be grabbed by a normal function.

\texttt{\textbackslash l\_peek\_token}

Token set by \texttt{\textbackslash peek\_after:Nw} and available for testing as described above.

\texttt{\textbackslash g\_peek\_token}

Token set by \texttt{\textbackslash peek\_gafter:Nw} and available for testing as described above.

\eqn{\texttt{\textbackslash peek\_catcode:NTF}}<\texttt{\textbackslash test\ token}\rangle \{\langle\texttt{\textbackslash true\ code}\rangle\} \{\langle\texttt{\textbackslash false\ code}\rangle\}

Tests if the next \langle\texttt{\textbackslash token}\rangle in the input stream has the same category code as the \langle\texttt{\textbackslash test\ token}\rangle (as defined by the test \texttt{\textbackslash token\_if\_eq\_catcode:NTF}). Spaces are respected by the test and the \langle\texttt{\textbackslash token}\rangle will be left in the input stream after the \langle\texttt{\textbackslash true\ code}\rangle or \langle\texttt{\textbackslash false\ code}\rangle (as appropriate to the result of the test).

\eqn{\texttt{\textbackslash peek\_catcode\_ignore\_spaces:NTF}}<\texttt{\textbackslash test\ token}\rangle \{\langle\texttt{\textbackslash true\ code}\rangle\} \{\langle\texttt{\textbackslash false\ code}\rangle\}

Tests if the next non-space \langle\texttt{\textbackslash token}\rangle in the input stream has the same category code as the \langle\texttt{\textbackslash test\ token}\rangle (as defined by the test \texttt{\textbackslash token\_if\_eq\_catcode:NTF}). Explicit and implicit space tokens (with character code 32 and category code 10) are ignored and removed by the test and the \langle\texttt{\textbackslash token}\rangle will be left in the input stream after the \langle\texttt{\textbackslash true\ code}\rangle or \langle\texttt{\textbackslash false\ code}\rangle (as appropriate to the result of the test).
\peek_catcode_remove:NTF \peek_catcode_remove:NTF ⟨test token⟩ \{⟨true code⟩\} \{⟨false code⟩\}

Tests if the next ⟨token⟩ in the input stream has the same category code as the ⟨test token⟩ (as defined by the test \token_if_eq_catcode:NNTF). Spaces are respected by the test and the ⟨token⟩ will be removed from the input stream if the test is true. The function will then place either the ⟨true code⟩ or ⟨false code⟩ in the input stream (as appropriate to the result of the test).

\peek_catcode_remove_ignore_spaces:NTF \peek_catcode_remove_ignore_spaces:NTF ⟨test token⟩ \{⟨true code⟩\} \{⟨false code⟩\}

Tests if the next non-space ⟨token⟩ in the input stream has the same category code as the ⟨test token⟩ (as defined by the test \token_if_eq_catcode:NNTF). Explicit and implicit space tokens (with character code 32 and category code 10) are ignored and removed by the test and the ⟨token⟩ will be removed from the input stream if the test is true. The function will then place either the ⟨true code⟩ or ⟨false code⟩ in the input stream (as appropriate to the result of the test).

\peek_charcode:NTF \peek_charcode:NTF ⟨test token⟩ \{⟨true code⟩\} \{⟨false code⟩\}

Tests if the next ⟨token⟩ in the input stream has the same character code as the ⟨test token⟩ (as defined by the test \token_if_eq_charcode:NNTF). Spaces are respected by the test and the ⟨token⟩ will be left in the input stream after the ⟨true code⟩ or ⟨false code⟩ (as appropriate to the result of the test).

\peek_charcode_ignore_spaces:NTF \peek_charcode_ignore_spaces:NTF ⟨test token⟩ \{⟨true code⟩\} \{⟨false code⟩\}

Tests if the next non-space ⟨token⟩ in the input stream has the same character code as the ⟨test token⟩ (as defined by the test \token_if_eq_charcode:NNTF). Explicit and implicit space tokens (with character code 32 and category code 10) are ignored and removed by the test and the ⟨token⟩ will be left in the input stream after the ⟨true code⟩ or ⟨false code⟩ (as appropriate to the result of the test).

\peek_charcode_remove:NTF \peek_charcode_remove:NTF ⟨test token⟩ \{⟨true code⟩\} \{⟨false code⟩\}

Tests if the next ⟨token⟩ in the input stream has the same character code as the ⟨test token⟩ (as defined by the test \token_if_eq_charcode:NNTF). Spaces are respected by the test and the ⟨token⟩ will be removed from the input stream if the test is true. The function will then place either the ⟨true code⟩ or ⟨false code⟩ in the input stream (as appropriate to the result of the test).
Tests if the next non-space ⟨token⟩ in the input stream has the same character code as the ⟨test token⟩ (as defined by the test \token_if_eq_charcode:NNTF). Explicit and implicit space tokens (with character code 32 and category code 10) are ignored and removed by the test and the ⟨token⟩ will be removed from the input stream if the test is true. The function will then place either the ⟨true code⟩ or ⟨false code⟩ in the input stream (as appropriate to the result of the test).

Tests if the next ⟨token⟩ in the input stream has the same meaning as the ⟨test token⟩ (as defined by the test \token_if_eq_meaning:NNTF). Spaces are respected by the test and the ⟨token⟩ will be left in the input stream after the ⟨true code⟩ or ⟨false code⟩ (as appropriate to the result of the test).

Tests if the next non-space ⟨token⟩ in the input stream has the same meaning as the ⟨test token⟩ (as defined by the test \token_if_eq_meaning:NNTF). Explicit and implicit space tokens (with character code 32 and category code 10) are ignored and removed by the test and the ⟨token⟩ will be removed from the input stream if the test is true. The function will then place either the ⟨true code⟩ or ⟨false code⟩ in the input stream (as appropriate to the result of the test).
7 Decomposing a macro definition

These functions decompose \TeX macros into their constituent parts: if the \langle token⟩ passed is not a macro then no decomposition can occur. In the later case, all three functions leave \scan_stop: in the input stream.

\verb|\token_get_arg_spec:N| \langle token⟩

If the \langle token⟩ is a macro, this function will leave the primitive \TeX argument specification in input stream as a string of tokens of category code 12 (with spaces having category code 10). Thus for example for a token \texttt{\next} defined by

\texttt{\cs_set:Npn \next #1#2 \{ x #1 y #2 \}}

will leave #1#2 in the input stream. If the \langle token⟩ is not a macro then \scan_stop: will be left in the input stream.

\textbf{\TeX Xhackers note:} If the arg spec. contains the string \texttt{-\rightarrow}, then the spec function will produce incorrect results.

\verb|\token_get_replacement_spec:N| \langle token⟩

If the \langle token⟩ is a macro, this function will leave the replacement text in input stream as a string of tokens of category code 12 (with spaces having category code 10). Thus for example for a token \texttt{\next} defined by

\texttt{\cs_set:Npn \next #1#2 \{ x \#1-y \#2 \}}

will leave x#1 y#2 in the input stream. If the \langle token⟩ is not a macro then \scan_stop: will be left in the input stream.

\textbf{\TeX Xhackers note:} If the arg spec. contains the string \texttt{-\rightarrow}, then the spec function will produce incorrect results.

\verb|\token_get_prefix_spec:N| \langle token⟩

If the \langle token⟩ is a macro, this function will leave the \TeX prefixes applicable in input stream as a string of tokens of category code 12 (with spaces having category code 10). Thus for example for a token \texttt{\next} defined by

\texttt{\cs_set:Npn \next #1\#2 \{ x \#1-y \#2 \}}

will leave \texttt{\long} in the input stream. If the \langle token⟩ is not a macro then \scan_stop: will be left in the input stream.
Part IX
The \texttt{l3int} package

Integers

Calculation and comparison of integer values can be carried out using literal numbers, \texttt{int}
registers, constants and integers stored in token list variables. The standard operators
\texttt{+}, \texttt{-}, \texttt{/} and \texttt{*} and parentheses can be used within such expressions to carry arithmetic
operations. This module carries out these functions on \textit{integer expressions} ("\texttt{intexpr}").

1 Integer expressions

\begin{verbatim}
\int_eval:n \{ \langle integer expression\rangle \}
\end{verbatim}

Evaluates the \langle integer expression\rangle, expanding any integer and token list variables within
the \langle expression\rangle to their content (without requiring \texttt{\int_use:N/\tl_use:N}) and applying
the standard mathematical rules. For example both
\begin{verbatim}
\int_eval:n \{ 5 + 4 * 3 - ( 3 + 4 * 5 ) \}
\end{verbatim}
and
\begin{verbatim}
\tl_new:N \l_my_tl
\tl_set:Nn \l_my_tl \{ 5 \}
\int_new:N \l_my_int
\int_set:Nn \l_my_int \{ 4 \}
\int_eval:n \{ \l_my_tl + \l_my_int * 3 - ( 3 + 4 * 5 ) \}
\end{verbatim}
both evaluate to \texttt{-6}. The \{\langle integer expression\rangle\} may contain the operators \texttt{+}, \texttt{-}, \texttt{*} and \texttt{/},
along with parenthesis ( and ). Any functions within the expressions should expand to an \langle integer denotation\rangle: a sequence of a sign and digits matching the regex \texttt{-?[0-9]+}).
After expansion \texttt{\int_eval:n} yields an \langle integer denotation\rangle which is left in the input stream.

\texttt{\TeXhackers} note: Exactly two expansions are needed to evaluate \texttt{\int_eval:n}. The result is \texttt{not} an \langle internal integer\rangle, and therefore requires suitable termination if used in a \TeX-
style integer assignment.

\begin{verbatim}
\int_abs:n \{ \langle integer expression\rangle \}
\end{verbatim}

Evaluates the \langle integer expression\rangle as described for \texttt{\int_eval:n} and leaves the absolute
value of the result in the input stream as an \langle integer denotation\rangle after two expansions.
\int\_div\_round:nn \{\textit{intexpr}_1\} \{\textit{intexpr}_2\}

Evaluates the two \textit{integer expressions} as described earlier, then divides the first value by the second, and rounds the result to the closest integer. Ties are rounded away from zero. Note that this is identical to using / directly in an \textit{integer expression}. The result is left in the input stream as an \textit{integer denotation} after two expansions.

\int\_div\_truncate:nn \{\textit{intexpr}_1\} \{\textit{intexpr}_2\}

Evaluates the two \textit{integer expressions} as described earlier, then divides the first value by the second, and rounds the result towards zero. Note that division using / rounds the result. The result is left in the input stream as an \textit{integer denotation} after two expansions.

\int\_max:nn \{\textit{intexpr}_1\} \{\textit{intexpr}_2\}
\int\_min:nn \{\textit{intexpr}_1\} \{\textit{intexpr}_2\}

Evaluates the \textit{integer expressions} as described for \int\_eval:n and leaves either the larger or smaller value in the input stream as an \textit{integer denotation} after two expansions.

\int\_mod:nn \{\textit{intexpr}_1\} \{\textit{intexpr}_2\}

Evaluates the two \textit{integer expressions} as described earlier, then calculates the integer remainder of dividing the first expression by the second. This is obtained by subtracting \int\_div\_truncate:nn \{\textit{intexpr}_1\} \{\textit{intexpr}_2\} times \textit{intexpr}_2 from \textit{intexpr}_1. Thus, the result has the same sign as \textit{intexpr}_1 and its absolute value is strictly less than that of \textit{intexpr}_2. The result is left in the input stream as an \textit{integer denotation} after two expansions.

2 Creating and initialising integers

\int\_new:N \{\textit{integer}\}
\int\_new:c

Creates a new \textit{integer} or raises an error if the name is already taken. The declaration is global. The \textit{integer} will initially be equal to 0.

\int\_const:Nn \{\textit{integer}\} \{\textit{integer expression}\}
\int\_const:cn

Creates a new constant \textit{integer} or raises an error if the name is already taken. The value of the \textit{integer} will be set globally to the \textit{integer expression}.

\int\_zero:N \{\textit{integer}\}
\int\_zero:c
\int\_gzero:N
\int\_gzero:c

Sets \textit{integer} to 0.
\int_zero_new:N \int_zero_new:c
\int_gzero_new:N \int_gzero_new:c

Ensures that the \textit{integer} exists globally by applying \int_new:N if necessary, then applies \int_(g)zero:N to leave the \textit{integer} set to zero.

\int_set_eq:NN \int_set_eq:cn \int_gset_eq:NN \int_gset_eq:cn

Sets the content of \textit{integer} equal to that of \textit{integer}.

\int_if_exist_p:N \int_if_exist:NTF \int_if_exist:N \int_if_exist:c \int_if_exist:TF \int_if_exist:c TF

Tests whether the \textit{int} is currently defined. This does not check that the \textit{int} really is an integer variable.

\int_add:Nn \int_add:cn \int_gadd:Nn \int_gadd:cn

Adds the result of the \textit{integer expression} to the current content of the \textit{integer}.

\int_decr:N \int_decr:c \int_gdecr:N \int_gdecr:c

Decreases the value stored in \textit{integer} by 1.

\int_incr:N \int_incr:c \int_gincr:N \int_gincr:c

Increases the value stored in \textit{integer} by 1.

\int_set:Nn \int_set:cn \int_gset:Nn \int_gset:cn

Sets \textit{integer} to the value of \textit{integer expression}, which must evaluate to an integer (as described for \int_eval:n).

3 Setting and incrementing integers
\int_{\text{sub:Nn}} \langle \text{integer} \rangle \{\langle \text{integer expression} \rangle\}

Subtracts the result of the \langle \text{integer expression} \rangle from the current content of the \langle \text{integer} \rangle.

Updated: 2011-10-22

4 Using integers

\int_{\text{use:N}} \langle \text{integer} \rangle

Recovers the content of an \langle \text{integer} \rangle and places it directly in the input stream. An error will be raised if the variable does not exist or if it is invalid. Can be omitted in places where an \langle \text{integer} \rangle is required (such as in the first and third arguments of \int_{\text{compare:nNnTF}}).

\texttt{\TeX}hackers note: \int_{\text{use:N}} is the \TeX{} primitive \texttt{\the}: this is one of several \LaTeX{}3 names for this primitive.

5 Integer expression conditionals

\int_{\text{compare:p:nNn}} \langle \text{intexpr}_1 \rangle \langle \text{relation} \rangle \langle \text{intexpr}_2 \rangle

\int_{\text{compare:nNnTF}} \langle \text{intexpr}_1 \rangle \langle \text{relation} \rangle \langle \text{intexpr}_2 \rangle

\{\langle \text{true code} \rangle\} \{\langle \text{false code} \rangle\}

This function first evaluates each of the \langle \text{integer expressions} \rangle as described for \int_{\text{-eval:n}}. The two results are then compared using the \langle \text{relation} \rangle:

\begin{align*}
\text{Equal} &= \text{=} \\
\text{Greater than} &= > \\
\text{Less than} &= <
\end{align*}
This function evaluates the \textit{(integer expressions)} as described for \texttt{\textbackslash int\_eval:n} and compares consecutive result using the corresponding \textit{(relation)}, namely it compares \texttt{\textbackslash intexpr\_1} and \texttt{\textbackslash intexpr\_2} using the \texttt{(relation\_1)}, then \texttt{\textbackslash intexpr\_2} and \texttt{\textbackslash intexpr\_3} using the \texttt{(relation\_2)}, until finally comparing \texttt{\textbackslash intexpr\_N} and \texttt{\textbackslash intexpr\_N+1} using the \texttt{(relation\_N)}. The test yields \texttt{true} if all comparisons are \texttt{true}. Each \textit{(integer expression)} is evaluated only once, and the evaluation is lazy, in the sense that if one comparison is \texttt{false}, then no other \textit{(integer expression)} is evaluated and no other comparison is performed. The \textit{(relations)} can be any of the following:

- Equal \texttt{=} or \texttt{==}
- Greater than or equal to \texttt{>=}
- Greater than \texttt{>}
- Less than or equal to \texttt{<=}
- Less than \texttt{<}
- Not equal \texttt{!=}
This function evaluates the \textit{test integer expression} and compares this in turn to each of the \textit{integer expression cases}. If the two are equal then the associated \textit{code} is left in the input stream. If any of the cases are matched, the \textit{(true code)} is also inserted into the input stream (after the code for the appropriate case), while if none match then the \textit{(false code)} is inserted. The function \texttt{\int_case:nn}, which does nothing if there is no match, is also available. For example
\[
{\int_case:nnF \{2 \times 5\}}
\]

\[
\{\{5\} \{\text{Small}\}\}, \{\{4 + 6\} \{\text{Medium}\}\}, \{\{-2 \times 10\} \{\text{Negative}\}\}, \{\text{No idea!}\}\}
\]

\texttt{\int_case:nn} will leave “Medium” in the input stream.

This function first evaluates the \textit{integer expression} as described for \texttt{\int_eval:n}. It then evaluates if this is odd or even, as appropriate.

\section{Integer expression loops}

\[
\int_do_until:nNnn \{\text{expr}_1\} \{\text{relation}\} \{\text{expr}_2\} \{\text{code}\}
\]
Places the \textit{code} in the input stream for \TeX{} to process, and then evaluates the relationship between the two \textit{integer expressions} as described for \texttt{\int_compare:nNnTF}. If the test is \texttt{false} then the \textit{code} will be inserted into the input stream again and a loop will occur until the \textit{relation} is \texttt{true}. 
\textbf{\int_do_while:nNnn} \ Star \ \textbf{\int_until_do:nNnn} \ Star

\textbf{\int_do_while:nNnn} \ \{\text{integer expression}\} \ \langle \text{relation} \rangle \ \{\text{integer expression}\} \ \{\text{code}\}

Places the \{code\} in the input stream for \TeX{} to process, and then evaluates the relationship between the two \{integer expressions\} as described for \textbf{\int_compare:nNnTF}. If the test is \texttt{true} then the \{code\} will be inserted into the input stream again and a loop will occur until the \{relation\} is \texttt{false}.

\textbf{\int_until_do:nNnn} \ Star \ \textbf{\int_while_do:nNnn} \ Star

\textbf{\int_until_do:nNnn} \ \{\text{integer expression}\} \ \langle \text{relation} \rangle \ \{\text{integer expression}\} \ \{\text{code}\}

Evaluates the relationship between the two \{integer expressions\} as described for \textbf{\int_compare:nNnTF}, and then places the \{code\} in the input stream if the \{relation\} is \texttt{false}. After the \{code\} has been processed by \TeX{} the test will be repeated, and a loop will occur until the test is \texttt{true}.

\textbf{\int_while_do:nNnn} \ Star \ \textbf{\int_do_until:nn} \ Star

\textbf{\int_while_do:nNnn} \ \{\text{integer expression}\} \ \langle \text{relation} \rangle \ \{\text{integer expression}\} \ \{\text{code}\}

Evaluates the relationship between the two \{integer expressions\} as described for \textbf{\int_compare:nNnTF}, and then places the \{code\} in the input stream if the \{relation\} is \texttt{true}. After the \{code\} has been processed by \TeX{} the test will be repeated, and a loop will occur until the test is \texttt{false}.

\textbf{\int_do_until:nn} \ Star \ \textbf{\int_do_while:nn} \ Star

\textbf{\int_do_until:nn} \ \{\text{integer relation}\} \ \{\text{code}\}

Places the \{code\} in the input stream for \TeX{} to process, and then evaluates the \{integer relation\} as described for \textbf{\int_compare:nNnTF}. If the test is \texttt{false} then the \{code\} will be inserted into the input stream again and a loop will occur until the \{relation\} is \texttt{true}.

\textbf{\int_do_while:nn} \ Star \ \textbf{\int_until_do:nn} \ Star

\textbf{\int_do_while:nn} \ \{\text{integer relation}\} \ \{\text{code}\}

Places the \{code\} in the input stream for \TeX{} to process, and then evaluates the \{integer relation\} as described for \textbf{\int_compare:nNnTF}. If the test is \texttt{true} then the \{code\} will be inserted into the input stream again and a loop will occur until the \{relation\} is \texttt{false}.

\textbf{\int_until_do:nn} \ Star \ \textbf{\int_while_do:nn} \ Star

\textbf{\int_until_do:nn} \ \{\text{integer relation}\} \ \{\text{code}\}

Evaluates the \{integer relation\} as described for \textbf{\int_compare:nNnTF}, and then places the \{code\} in the input stream if the \{relation\} is \texttt{false}. After the \{code\} has been processed by \TeX{} the test will be repeated, and a loop will occur until the test is \texttt{true}.

\textbf{\int_while_do:nn} \ Star

\textbf{\int_while_do:nn} \ \{\text{integer relation}\} \ \{\text{code}\}

Evaluates the \{integer relation\} as described for \textbf{\int_compare:nNnTF}, and then places the \{code\} in the input stream if the \{relation\} is \texttt{true}. After the \{code\} has been processed by \TeX{} the test will be repeated, and a loop will occur until the test is \texttt{false}. Updated: 2013-01-13
7 Integer step functions

This function first evaluates the *initial value*, *step* and *final value*, all of which should be integer expressions. The *(function)* is then placed in front of each *value* from the *(initial value)* to the *(final value)* in turn (using *(step)* between each *value*). The *(step)* must be non-zero. If the *(step)* is positive, the loop stops when the *(value)* becomes larger than the *(final value)*. If the *(step)* is negative, the loop stops when the *(value)* becomes smaller than the *(final value)*. The *(function)* should absorb one numerical argument. For example

```latex
\cs_set:Npn \my_func:n #1 { [I~saw~#1] \quad }
\int_step_function:nnnN { 1 } { 1 } { 5 } \my_func:n
```

would print

[I saw 1] [I saw 2] [I saw 3] [I saw 4] [I saw 5]

8 Formatting integers

Integers can be placed into the output stream with formatting. These conversions apply to any integer expressions.

```latex
\int_to_arabic:n \{integer expression\}
```

Places the value of the *(integer expression)* in the input stream as digits, with category code 12 (other).
\int_to_alph:n \{\langle\text{integer expression}\rangle\}\n
Evaluates the \langle\text{integer expression}\rangle\ and converts the result into a series of letters, which are then left in the input stream. The conversion rule uses the 26 letters of the English alphabet, in order, adding letters when necessary to increase the total possible range of representable numbers. Thus

\int_to_alph:n \{ 1 \}

places a in the input stream,

\int_to_alph:n \{ 26 \}

is represented as z and

\int_to_alph:n \{ 27 \}

is converted to aa. For conversions using other alphabets, use \int_to_symbols:nnn to define an alphabet-specific function. The basic \int_to_alph:n and \int_to_Alph:n functions should not be modified. The resulting tokens are digits with category code 12 (other) and letters with category code 11 (letter).

\int_to_symbols:nnn \{\langle\text{integer expression}\rangle\} \{\langle\text{total symbols}\rangle\}\n
\langle\text{value to symbol mapping}\rangle

This is the low-level function for conversion of an \langle\text{integer expression}\rangle\ into a symbolic form (which will often be letters). The \langle\text{total symbols}\rangle\ available should be given as an integer expression. Values are actually converted to symbols according to the \langle\text{value to symbol mapping}\rangle. This should be given as \langle\text{total symbols}\rangle\ pairs of entries, a number and the appropriate symbol. Thus the \int_to_alph:n function is defined as

\cs_new:Npn \int_to_alph:n \#1
\{\}
\int_to_symbols:nnn \{\#1\} \{ 26 \}\n\{\}
\{ 1 \} \{ a \}\n\{ 2 \} \{ b \}\n\ldots\n\{ 26 \} \{ z \}\n\}
\}

\int_to_bin:n \{\langle\text{integer expression}\rangle\}\n
Calculates the value of the \langle\text{integer expression}\rangle\ and places the binary representation of the result in the input stream.
\int_to_hex:n \langle integer expression \rangle
\int_to_Hex:n \langle integer expression \rangle
\int_to_oct:n \langle integer expression \rangle
\int_to_base:nn \langle integer expression \rangle \{ base \}
\int_to_Roman:n \langle integer expression \rangle
\int_to_Roman:n \langle integer expression \rangle
\int_from_alph:n \langle letters \rangle
\int_from_bin:n \langle binary number \rangle

Calculates the value of the \langle integer expression \rangle and places the hexadecimal (base 16) representation of the result in the input stream. Letters are used for digits beyond 9: lower case letters for \int_to_hex:n and upper case ones for \int_to_Hex:n. The resulting tokens are digits with category code 12 (other) and letters with category code 11 (letter).

Calculates the value of the \langle integer expression \rangle and places the octal (base 8) representation of the result in the input stream. The resulting tokens are digits with category code 12 (other) and letters with category code 11 (letter).

Calculates the value of the \langle integer expression \rangle and converts it into the appropriate representation in the \langle base \rangle; the later may be given as an integer expression. For bases greater than 10 the higher “digits” are represented by letters from the English alphabet: lower case letters for \int_to_base:n and upper case ones for \int_to_Base:n. The maximum \langle base \rangle value is 36. The resulting tokens are digits with category code 12 (other) and letters with category code 11 (letter).

\textbf{\TeXhacks note:} This is a generic version of \int_to_bin:n, etc.

Places the value of the \langle integer expression \rangle in the input stream as Roman numerals, either lower case (\int_to_roman:n) or upper case (\int_to_Roman:n). The Roman numerals are letters with category code 11 (letter).

\section{Converting from other formats to integers}

Converts the \langle letters \rangle into the integer (base 10) representation and leaves this in the input stream. The \langle letters \rangle are first converted to a string, with no expansion. Lower and upper case letters from the English alphabet may be used, with “a” equal to 1 through to “z” equal to 26. The function also accepts a leading sign, made of + and -. This is the inverse function of \int_to_alph:n and \int_to_Alph:n.

Converts the \langle binary number \rangle into the integer (base 10) representation and leaves this in the input stream. The \langle binary number \rangle is first converted to a string, with no expansion. The function accepts a leading sign, made of + and -, followed by binary digits. This is the inverse function of \int_to_bin:n.
\int_from_hex:n \{\textit{hexadecimal number}\}

Converts the \textit{hexadecimal number} into the integer (base 10) representation and leaves this in the input stream. Digits greater than 9 may be represented in the \textit{hexadecimal number} by upper or lower case letters. The \textit{hexadecimal number} is first converted to a string, with no expansion. The function also accepts a leading sign, made of + and -. This is the inverse function of \int_to_hex:n and \int_to_Hex:n.

\int_from_oct:n \{\textit{octal number}\}

Converts the \textit{octal number} into the integer (base 10) representation and leaves this in the input stream. The \textit{octal number} is first converted to a string, with no expansion. The function accepts a leading sign, made of + and -, followed by octal digits. This is the inverse function of \int_to_oct:n.

\int_from_roman:n \{\textit{roman numeral}\}

Converts the \textit{roman numeral} into the integer (base 10) representation and leaves this in the input stream. The \textit{roman numeral} is first converted to a string, with no expansion. The \textit{roman numeral} may be in upper or lower case; if the numeral contains characters besides mdclxvi or MDCLXVI then the resulting value will be $-1$. This is the inverse function of \int_to_roman:n and \int_to_Roman:n.

\int_from_base:nn \{\textit{number}\} \{(base)\}

Converts the \textit{number} expressed in \textit{base} into the appropriate value in base 10. The \textit{number} is first converted to a string, with no expansion. The \textit{number} should consist of digits and letters (either lower or upper case), plus optionally a leading sign. The maximum \textit{base} value is 36. This is the inverse function of \int_to_base:nn and \int_to_Base:nn.

## 10 Viewing integers

\int_show:N \{\textit{integer}\}

\int_show:C

Displays the value of the \textit{integer} on the terminal.

\int_show:n \{\textit{integer expression}\}

Displays the result of evaluating the \textit{integer expression} on the terminal.
11 Constant integers

Integer values used with primitive tests and assignments: self-terminating nature makes these more convenient and faster than literal numbers.

\c_minus_one  \c_zero  \c_one  \c_two  \c_three  \c_four  \c_five  \c_six  \c_seven  \c_eight  \c_nine  \c_ten  \c_eleven  \c_twelve  \c_thirteen  \c_fourteen  \c_fifteen  \c_sixteen  \c_thirty_two  \c_one_hundred  \c_two_hundred_fifty_five  \c_two_hundred_fifty_six  \c_one_thousand  \c_ten_thousand

\c_max_int

The maximum value that can be stored as an integer.

\c_max_register_int

Maximum number of registers.

12 Scratch integers

\l_tmpa_int  \l_tmpb_int

Scratch integer for local assignment. These are never used by the kernel code, and so are safe for use with any \TeX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

\g_tmpa_int  \g_tmpb_int

Scratch integer for global assignment. These are never used by the kernel code, and so are safe for use with any \TeX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.
13 Primitive conditionals

\texttt{\textbackslash if\_int\_compare:w} * \texttt{\textbackslash if\_int\_compare:w (integer) (relation) (integer)}
\texttt{(true code)}
\texttt{\textbackslash else:}
\texttt{(false code)}
\texttt{\textbackslash fi:}

Compare two integers using \langle relation \rangle, which must be one of \(=\), \(<\) or \(>\) with category code 12. The \texttt{\textbackslash else:} branch is optional.

\textbf{\TeX hack\textbackslash hackers note:} These are both names for the \TeX primitive \texttt{\textbackslash ifnum}.

\texttt{\textbackslash if\_case:w} * \texttt{\textbackslash if\_case:w (integer) (case)}
\texttt{\or: (case)\or: ...
\texttt{\textbackslash else: (default)}
\texttt{\textbackslash fi:}

Selects a case to execute based on the value of the \langle integer \rangle. The first case (\langle case0 \rangle) is executed if \langle integer \rangle is 0, the second (\langle case1 \rangle) if the \langle integer \rangle is 1, etc. The \langle integer \rangle may be a literal, a constant or an integer expression (\textit{e.g.} using \texttt{\textbackslash int\_eval:n}).

\textbf{\TeX hack\textbackslash hackers note:} These are the \TeX primitives \texttt{\textbackslash ifcase} and \texttt{\or}.

\texttt{\textbackslash if\_int\_odd:w} * \texttt{\textbackslash if\_int\_odd:w (tokens) (optional space)}
\texttt{(true code)}
\texttt{\textbackslash else:}
\texttt{(true code)}
\texttt{\textbackslash fi:}

Expands \langle tokens \rangle until a non-numeric token or a space is found, and tests whether the resulting \langle integer \rangle is odd. If so, \texttt{(true code)} is executed. The \texttt{\textbackslash else:} branch is optional.

\textbf{\TeX hack\textbackslash hackers note:} This is the \TeX primitive \texttt{\ifodd}.

14 Internal functions

\texttt{\_\_int\_to\_roman:w} * \texttt{\_\_int\_to\_roman:w (integer) (space) or (non-expandable token)}

Converts \langle integer \rangle to its lower case Roman representation. Expansion ends when a space or non-expandable token is found. Note that this function produces a string of letters with category code 12 and that protected functions \textit{are} expanded by this process. Negative \langle integer \rangle values result in no output, although the function does not terminate expansion until a suitable endpoint is found in the same way as for positive numbers.

\textbf{\TeX hack\textbackslash hackers note:} This is the \TeX primitive \texttt{\romannumeral} renamed.
Expands \( \text{tokens} \) until an \( \text{integer} \) is formed. One space may be gobbled in the process.

\textbf{\TeXhackers note:} This is the \TeX primitive \texttt{\number}.

Evaluates \( \text{integer expression} \) as described for \texttt{\int_eval:n}. The evaluation stops when an unexpandable token which is not a valid part of an integer is read or when \texttt{\__int_eval_end:} is reached. The latter is gobbled by the scanner mechanism: \texttt{\__int_eval_end:} itself is unexpandable but used correctly the entire construct is expandable.

\textbf{\TeXhackers note:} This is the \TeX primitive \texttt{\numexpr}.

These are used within \texttt{\int_compare:n(TF)}, \texttt{\dim_compare:n(TF)} and so on to recover correctly if the \texttt{n}-type argument does not contain a properly-formed relation.
Part X

The l3skip package

Dimensions and skips

\LaTeX{}3 provides two general length variables: \texttt{dim} and \texttt{skip}. Lengths stored as \texttt{dim} variables have a fixed length, whereas \texttt{skip} lengths have a rubber (stretch/shrink) component. In addition, the \texttt{muskip} type is available for use in math mode: this is a special form of \texttt{skip} where the lengths involved are determined by the current math font (in \texttt{mu}). There are common features in the creation and setting of length variables, but for clarity the functions are grouped by variable type.

1 Creating and initialising \texttt{dim} variables

\begin{verbatim}
\dim_new:N \dim_new:c
\dim_new:N \dim_new:c

\dim_new:N \dim_new:c
\dim_new:N \dim_new:c

\dim_new:N \dim_new:c
\dim_new:N \dim_new:c
\dim_new:N \dim_new:c
\dim_new:N \dim_new:c
\end{verbatim}

Creates a new \texttt{dimension} or raises an error if the name is already taken. The declaration is global. The \texttt{dimension} will initially be equal to \texttt{0pt}.

\begin{verbatim}
\dim_const:Nn \dim_const:cn
\dim_const:Nn \dim_const:cn
\end{verbatim}

Creates a new constant \texttt{dimension} or raises an error if the name is already taken. The value of the \texttt{dimension} will be set globally to the \texttt{dimension expression}.

\begin{verbatim}
\dim_zero:N \dim_zero:c
\dim_zero:N \dim_zero:c
\dim_zero:N \dim_zero:c
\dim_zero:N \dim_zero:c
\end{verbatim}

Sets \texttt{dimension} to \texttt{0pt}.

\begin{verbatim}
\dim_zero_new:N \dim_zero_new:c
\dim_zero_new:N \dim_zero_new:c
\dim_zero_new:N \dim_zero_new:c
\dim_zero_new:N \dim_zero_new:c
\end{verbatim}

Ensures that the \texttt{dimension} exists globally by applying \texttt{dim_new:N} if necessary, then applies \texttt{dim_(g)zero:N} to leave the \texttt{dimension} set to zero.

\begin{verbatim}
\dim_if_exist_p:N \dim_if_exist_p:c
\dim_if_exist_p:N \dim_if_exist_p:c
\dim_if_exist:N \dim_if_exist:c\texttt{TF}
\dim_if_exist:N \dim_if_exist:c\texttt{TF}
\end{verbatim}

Tests whether the \texttt{dimension} is currently defined. This does not check that the \texttt{dimension} really is a dimension variable.
2 Setting dim variables

\dim_add:Nn \langle \text{dimension} \rangle \{\langle \text{dimension expression} \rangle\}

Adds the result of the \langle \text{dimension expression} \rangle to the current content of the \langle \text{dimension} \rangle.

\dim_set:Nn \langle \text{dimension} \rangle \{\langle \text{dimension expression} \rangle\}

Sets \langle \text{dimension} \rangle to the value of \langle \text{dimension expression} \rangle, which must evaluate to a length with units.

\dim_set_eq:NN \langle \text{dimension} \rangle_1 \langle \text{dimension} \rangle_2

Sets the content of \langle \text{dimension} \rangle_1 equal to that of \langle \text{dimension} \rangle_2.

\dim_sub:Nn \langle \text{dimension} \rangle \{\langle \text{dimension expression} \rangle\}

Subtracts the result of the \langle \text{dimension expression} \rangle from the current content of the \langle \text{dimension} \rangle.

3 Utilities for dimension calculations

\dim_abs:n \langle \text{dimexpr} \rangle

Converts the \langle \text{dimexpr} \rangle to its absolute value, leaving the result in the input stream as a \langle \text{dimension denotation} \rangle.

\dim_max:nn \langle \text{dimexpr}_1 \rangle \{\langle \text{dimexpr}_2 \rangle\}
\dim_min:nn \langle \text{dimexpr}_1 \rangle \{\langle \text{dimexpr}_2 \rangle\}

Evaluates the two \langle \text{dimension expressions} \rangle and leaves either the maximum or minimum value in the input stream as appropriate, as a \langle \text{dimension denotation} \rangle.
\dim_ratio:nn \dim_ratio:nn \{\dimexpr_1\} \{\dimexpr_2\}

Parses the two \textit{dimension expressions} and converts the ratio of the two to a form suitable for use inside a \textit{dimension expression}. This ratio is then left in the input stream, allowing syntax such as

\begin{verbatim}
\dim_set:Nn \l_my_dim
  { 10 pt * \dim_ratio:nn \{ 5 pt \} \{ 10 pt \} }
\end{verbatim}

The output of \texttt{\dim_ratio:nn} on full expansion is a ratio expression between two integers, with all distances converted to scaled points. Thus

\begin{verbatim}
\tl_set:Nx \l_my_tl \{ \dim_ratio:nn \{ 5 pt \} \{ 10 pt \} }
\tl_show:N \l_my_tl
\end{verbatim}

will display $327680/655360$ on the terminal.

\section{Dimension expression conditionals}

\dim_compare_p:nNn \dim_compare_p:nNn \{\dimexpr_1\} \langle\text{relation}\rangle \{\dimexpr_2\}

\dim_compare:nNnTF \dim_compare:nNnTF \{\dimexpr_1\} \langle\text{relation}\rangle \{\dimexpr_2\}
\{\langle\text{true code}\rangle\} \{\langle\text{false code}\rangle\}

This function first evaluates each of the \textit{dimension expressions} as described for \texttt{\dim_eval:n}. The two results are then compared using the \textit{relation}:

\begin{center}
Equal \quad = \\
Greater than \quad > \\
Less than \quad <
\end{center}
This function evaluates the (dimension expressions) as described for \texttt{\verbatimverbatim{dim_eval:n}} and compares consecutive result using the corresponding (relation), namely it compares \texttt{(dimexpr}_1\texttt{)} and \texttt{(dimexpr}_2\texttt{)} using the (relation$_1$), then \texttt{(dimexpr}_2\texttt{)} and \texttt{(dimexpr}_3\texttt{)} using the (relation$_2$), until finally comparing \texttt{(dimexpr}_N\texttt{)} and \texttt{(dimexpr}_N+1\texttt{)} using the (relation$_N$). The test yields \texttt{true} if all comparisons are \texttt{true}. Each (dimension expression) is evaluated only once, and the evaluation is lazy, in the sense that if one comparison is \texttt{false}, then no other (dimension expression) is evaluated and no other comparison is performed. The (relations) can be any of the following:

- Equal \texttt{=} or \texttt{==}
- Greater than or equal to \texttt{>=}
- Greater than \texttt{>}
- Less than or equal to \texttt{<=}
- Less than \texttt{<}
- Not equal \texttt{!=}
\dim_case:nnTF
\dim_case:nnTF {⟨test dimension expression⟩}
{
  {⟨dimexpr case1⟩} {⟨code case1⟩}
  {⟨dimexpr case2⟩} {⟨code case2⟩}
  ...
  {⟨dimexpr case_n⟩} {⟨code case_n⟩}
}{⟨true code⟩}
{⟨false code⟩}

This function evaluates the ⟨test dimension expression⟩ and compares this in turn to each of the ⟨dimension expression cases⟩. If the two are equal then the associated ⟨code⟩ is left in the input stream. If any of the cases are matched, the ⟨true code⟩ is also inserted into the input stream (after the code for the appropriate case), while if none match then the ⟨false code⟩ is inserted. The function \dim_case:nn, which does nothing if there is no match, is also available. For example
\dim_set:Nn \l_tmpa_dim { 5 pt }
\dim_case:nnF
  { 2 \l_tmpa_dim }
  {{ 5 pt } { Small } }
  { 4 pt + 6 pt } { Medium } 
  { - 10 pt } { Negative } 
  { No idea! }

will leave “Medium” in the input stream.

5 Dimension expression loops

\dim_do_until:nNnn
\dim_do_until:nNnn {⟨dimexpr1⟩} ⟨relation⟩ {⟨dimexpr2⟩} {⟨code⟩}

Places the ⟨code⟩ in the input stream for TeX to process, and then evaluates the relationship between the two ⟨dimension expressions⟩ as described for \dim_compare:nNnTF. If the test is false then the ⟨code⟩ will be inserted into the input stream again and a loop will occur until the ⟨relation⟩ is true.

\dim_do_while:nNnn
\dim_do_while:nNnn {⟨dimexpr1⟩} ⟨relation⟩ {⟨dimexpr2⟩} {⟨code⟩}

Places the ⟨code⟩ in the input stream for TeX to process, and then evaluates the relationship between the two ⟨dimension expressions⟩ as described for \dim_compare:nNnTF. If the test is true then the ⟨code⟩ will be inserted into the input stream again and a loop will occur until the ⟨relation⟩ is false.
Evaluates the relationship between the two (dimension expressions) as described for \dim_compare:nTF, and then places the (code) in the input stream if the (relation) is false. After the (code) has been processed by \TeX{} the test will be repeated, and a loop will occur until the test is true.

Evaluates the relationship between the two (dimension expressions) as described for \dim_compare:nTF, and then places the (code) in the input stream if the (relation) is true. After the (code) has been processed by \TeX{} the test will be repeated, and a loop will occur until the test is false.

Places the (code) in the input stream for \TeX{} to process, and then evaluates the (dimension relation) as described for \dim_compare:nTF. If the test is false then the (code) will be inserted into the input stream again and a loop will occur until the (relation) is true.

Places the (code) in the input stream for \TeX{} to process, and then evaluates the (dimension relation) as described for \dim_compare:nTF. If the test is true then the (code) will be inserted into the input stream again and a loop will occur until the (relation) is false.

Evaluates the (dimension relation) as described for \dim_compare:nTF, and then places the (code) in the input stream if the (relation) is false. After the (code) has been processed by \TeX{} the test will be repeated, and a loop will occur until the test is true.

Evaluates the (dimension relation) as described for \dim_compare:nTF, and then places the (code) in the input stream if the (relation) is true. After the (code) has been processed by \TeX{} the test will be repeated, and a loop will occur until the test is false.

6 Using dim expressions and variables

Evaluates the (dimension expression), expanding any dimensions and token list variables within the (expression) to their content (without requiring \dim_use:N/\tl_use:N) and applying the standard mathematical rules. The result of the calculation is left in the input stream as a (dimension denotation) after two expansions. This will be expressed in points (pt), and will require suitable termination if used in a \TeX{}-style assignment as it is not an (internal dimension).
\dim_use:N \dim_use:N \langle \text{dimension} \rangle
\dim_use:c \dim_use:c

Recovers the content of a \langle \text{dimension} \rangle and places it directly in the input stream. An error will be raised if the variable does not exist or if it is invalid. Can be omitted in places where a \langle \text{dimension} \rangle is required (such as in the argument of \dim_eval:n).

\TeX\text{hackers note:} \dim_use:N is the \TeX\ primitive \the: this is one of several \LaTeX3 names for this primitive.

\dim_to_decimal:n \dim_to_decimal:n \{ \langle \text{dimexpr} \rangle \}

Evaluates the \langle \text{dimension expression} \rangle, and leaves the result, expressed in points (pt) in the input stream, with \textit{no units}. The result is rounded by \TeX\ to four or five decimal places. If the decimal part of the result is zero, it is omitted, together with the decimal marker.

For example

\dim_to_decimal:n \{ 1bp \}

leaves 1.00374 in the input stream, \textit{i.e.} the magnitude of one “big point” when converted to \TeX\ points.

\dim_to_decimal_in_bp:n \dim_to_decimal_in_bp:n \{ \langle \text{dimexpr} \rangle \}

Evaluates the \langle \text{dimension expression} \rangle, and leaves the result, expressed in big points (bp) in the input stream, \textit{with no units}. The result is rounded by \TeX\ to four or five decimal places. If the decimal part of the result is zero, it is omitted, together with the decimal marker.

For example

\dim_to_decimal_in_bp:n \{ 1pt \}

leaves 0.99628 in the input stream, \textit{i.e.} the magnitude of one \TeX\ point when converted to big points.

\dim_to_decimal_in_unit:nn \dim_to_decimal_in_unit:nn \{ \langle \text{dimexpr} _1 \rangle \} \{ \langle \text{dimexpr} _2 \rangle \}

Evaluates the \langle \text{dimension expressions} \rangle, and leaves the value of \langle \text{dimexpr} _1 \rangle, expressed in a unit given by \langle \text{dimexpr} _2 \rangle, in the input stream. The result is a decimal number, rounded by \TeX\ to four or five decimal places. If the decimal part of the result is zero, it is omitted, together with the decimal marker.

For example

\dim_to_decimal_in_unit:nn \{ 1bp \} \{ 1mm \}

leaves 0.35277 in the input stream, \textit{i.e.} the magnitude of one big point when converted to millimetres.
\texttt{\dim_to_fp:n \{\texttt{dimexpr}\}}

Expands to an internal floating point number equal to the value of the \texttt{\texttt{dimexpr}} in pt. Since dimension expressions are evaluated much faster than their floating point equivalent, \texttt{\dim_to_fp:n} can be used to speed up parts of a computation where a low precision is acceptable.

7 Viewing dim variables

\texttt{\dim_show:N \{\texttt{dimension}\}}

\texttt{\dim_show:c \{\texttt{dimension}\}}

Displays the value of the \texttt{\texttt{dimension}} on the terminal.

\texttt{\dim_show:n \{\texttt{dimension expression}\}}

\texttt{\dim_show:n \{\texttt{dimension expression}\}}

Displays the result of evaluating the \texttt{\texttt{dimension expression}} on the terminal.

8 Constant dimensions

\texttt{\c_max_dim}

The maximum value that can be stored as a dimension. This can also be used as a component of a skip.

\texttt{\c_zero_dim}

A zero length as a dimension. This can also be used as a component of a skip.

9 Scratch dimensions

\texttt{\l_tmpa_dim \ l_tmpb_dim}

Scratch dimension for local assignment. These are never used by the kernel code, and so are safe for use with any \LaTeX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

\texttt{\g_tmpa_dim \ g_tmpb_dim}

Scratch dimension for global assignment. These are never used by the kernel code, and so are safe for use with any \LaTeX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.
10 Creating and initialising skip variables

\skip_new:N \skip_new:c
Creates a new \langle skip \rangle or raises an error if the name is already taken. The declaration is global. The \langle skip \rangle will initially be equal to 0 pt.

\skip_const:Nn \skip_const:cn
\skip_set:Nn \skip_set:cn
\skip_add:Nn \skip_add:cn
\skip_gadd:Nn \skip_gadd:cn
\skip_zero:N \skip_zero:c
\skip_gzero:N \skip_gzero:c
\skip_zero_new:N \skip_zero_new:c
\skip_gzero_new:N \skip_gzero_new:c
Ensures that the \langle skip \rangle exists globally by applying \skip_new:N if necessary, then applies \skip_(g)zero:N to leave the \langle skip \rangle set to zero.

\skip_if_exist_p:N \skip_if_exist_p:cn \skip_if_exist:NTF \skip_if_exist:cTF
Tests whether the \langle skip \rangle is currently defined. This does not check that the \langle skip \rangle really is a skip variable.

11 Setting skip variables

\skip_add:Nn \skip_add:cn
\skip_set:Nn \skip_set:cn
\skip_gadd:Nn \skip_gadd:cn
\skip_gset:Nn \skip_gset:cn
Adds the result of the \langle skip expression \rangle to the current content of the \langle skip \rangle.
Sets \langle skip \rangle to the value of \langle skip expression \rangle, which must evaluate to a length with units and may include a rubber component (for example 1 cm plus 0.5 cm).
\skip_set_eq:NN \skip_set_eq:(cN|Nc|cc) \skip_gset_eq:NN \skip_gset_eq:(cN|Nc|cc)

Sets the content of \langle \text{skip} \rangle_1 equal to that of \langle \text{skip} \rangle_2.

\skip_sub:Nn \skip_sub:cn \skip_gsub:Nn \skip_gsub:cn

Subtracts the result of the \langle \text{skip expression} \rangle from the current content of the \langle \text{skip} \rangle.

12 Skip expression conditionals

\skip_if_eq_p:nn \skip_if_eq:nnF \dim_compare:nTF

This function first evaluates each of the \langle \text{skip expressions} \rangle as described for \texttt{\skip_eval:n}. The two results are then compared for exact equality, \textit{i.e.} both the fixed and rubber components must be the same for the test to be true.

\skip_if_finite_p:n \skip_if_finite:nTF

Evaluates the \langle \text{skip expression} \rangle as described for \texttt{\skip_eval:n}, and then tests if all of its components are finite.

13 Using skip expressions and variables

\skip_eval:n \skip_eval:nn \skip_eval:n \langle \text{skip expression} \rangle

Evaluates the \langle \text{skip expression} \rangle, expanding any skips and token list variables within the \langle \text{expression} \rangle to their content (without requiring \texttt{\skip_use:N/\tl_use:N}) and applying the standard mathematical rules. The result of the calculation is left in the input stream as a \langle \text{glue denotation} \rangle after two expansions. This will be expressed in points (pt), and will require suitable termination if used in a \TeX-style assignment as it is \textit{not an \langle \text{internal glue} \rangle}. 
\skip_use:N \skip_use:c

Recovers the content of a \textit{skip} and places it directly in the input stream. An error will be raised if the variable does not exist or if it is invalid. Can be omitted in places where a \textit{dimension} is required (such as in the argument of \texttt{\skip_eval:n}).

\textbf{\texttt{\TeX}hackers note:} \texttt{\skip_use:N} is the \texttt{\TeX} primitive \texttt{\the}: this is one of several \texttt{\expandafter\\\TeX3} names for this primitive.

### 14 Viewing \texttt{\skip} variables

\texttt{\skip_show:N} \texttt{\skip_show:c}

Displays the value of the \textit{skip} on the terminal.

\texttt{\skip_show:n} {\texttt{\langle skip expression\rangle}}

Displays the result of evaluating the \textit{skip expression} on the terminal.

### 15 Constant skips

\texttt{\c_max_skip}

The maximum value that can be stored as a \texttt{\skip} (equal to \texttt{\c_max_dim} in length), with no stretch nor shrink component.

\texttt{\c_zero_skip}

A zero length as a \texttt{\skip}, with no stretch nor shrink component.

### 16 Scratch skips

\texttt{\l_tmpa_skip} \texttt{\l_tmpb_skip}

Scratch \texttt{\skip} for local assignment. These are never used by the kernel code, and so are safe for use with any \texttt{\expandafter\\\TeX3}-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

\texttt{\g_tmpa_skip} \texttt{\g_tmpb_skip}

Scratch \texttt{\skip} for global assignment. These are never used by the kernel code, and so are safe for use with any \texttt{\expandafter\\\TeX3}-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.
17 Inserting skips into the output

\skip_horizontal:N \skip_horizontal:N \skip_horizontal:n\{\langle\text{skipexpr}\rangle\}

Inserts a horizontal \langle\text{skip}\rangle into the current list.

\TeXhackers\ note: \skip_horizontal:N is the\ TeX\ primitive \hskip\ renamed.

\skip_vertical:N \skip_vertical:n\{\langle\text{skipexpr}\rangle\}

Inserts a vertical \langle\text{skip}\rangle into the current list.

\TeXhackers\ note: \skip_vertical:N is the\ TeX\ primitive \vskip\ renamed.

18 Creating and initialising \texttt{muskip} variables

\muskip_new:N \muskip_new:c

Creates a new \langle\text{muskip}\rangle or raises an error if the name is already taken. The declaration is global. The \langle\text{muskip}\rangle will initially be equal to 0\textmu .

\muskip_const:Nn \muskip_const:cn

New: 2012-03-05

Creates a new constant \langle\text{muskip}\rangle or raises an error if the name is already taken. The value of the \langle\text{muskip}\rangle will be set globally to the \langle\text{muskip expression}\rangle.

\muskip_zero:N \muskip_zero:c \muskip_gzero:N \muskip_gzero:c

\muskip_zero_new:N \muskip_zero_new:c \muskip_gzero_new:N \muskip_gzero_new:c

New: 2012-01-07

Ensures that the \langle\text{muskip}\rangle exists globally by applying \muskip_new:N if necessary, then applies \muskip_{\text{(g)zero}}:N to leave the \langle\text{muskip}\rangle set to zero.

\muskip_if_exist_p:N \muskip_if_exist_p:c \muskip_if_exist:NTF \muskip_if_exist:ctF

New: 2012-03-03

Tests whether the \langle\text{muskip}\rangle is currently defined. This does not check that the \langle\text{muskip}\rangle really is a muskip variable.

88
19 Setting \texttt{muskip} variables

\begin{verbatim}
\texttt{\textbackslash muskip\_add:Nn} \texttt{\textbackslash muskip\_add:cn} \\
\texttt{\textbackslash muskip\_gadd:Nn} \texttt{\textbackslash muskip\_gadd:cn} \texttt{Updated: 2011-10-22}
\end{verbatim}

\begin{verbatim}
\texttt{\textbackslash muskip\_set:Nn} \texttt{\textbackslash muskip\_set:cn} \\
\texttt{\textbackslash muskip\_gset:Nn} \texttt{\textbackslash muskip\_gset:cn} \texttt{Updated: 2011-10-22}
\end{verbatim}

\begin{verbatim}
\texttt{\textbackslash muskip\_set\_eq:NN} \texttt{\textbackslash muskip\_set\_eq:\texttt{(cN|Nc|cc)}} \\
\texttt{\textbackslash muskip\_gset\_eq:NN} \texttt{\textbackslash muskip\_gset\_eq:\texttt{(cN|Nc|cc)}} \texttt{Updated: 2011-10-22}
\end{verbatim}

\begin{verbatim}
\texttt{\textbackslash muskip\_sub:Nn} \texttt{\textbackslash muskip\_sub:cn} \\
\texttt{\textbackslash muskip\_gsub:Nn} \texttt{\textbackslash muskip\_gsub:cn} \texttt{Updated: 2011-10-22}
\end{verbatim}

19 Setting \texttt{muskip} variables

\begin{verbatim}
\texttt{\textbackslash muskip\_add:Nn} \langle \texttt{muskip} \rangle \{\langle \texttt{muskip expression} \rangle \}
\end{verbatim}

Adds the result of the \langle \texttt{muskip expression} \rangle to the current content of the \langle \texttt{muskip} \rangle.

\begin{verbatim}
\texttt{\textbackslash muskip\_set:Nn} \langle \texttt{muskip} \rangle \{\langle \texttt{muskip expression} \rangle \}
\end{verbatim}

Sets \langle \texttt{muskip} \rangle to the value of \langle \texttt{muskip expression} \rangle, which must evaluate to a math length with units and may include a rubber component (for example 1 \texttt{mu} plus 0.5 \texttt{mu}.

\begin{verbatim}
\texttt{\textbackslash muskip\_set\_eq:NN} \langle \texttt{muskip_1} \rangle \langle \texttt{muskip_2} \rangle
\end{verbatim}

Sets the content of \langle \texttt{muskip_1} \rangle equal to that of \langle \texttt{muskip_2} \rangle.

\begin{verbatim}
\texttt{\textbackslash muskip\_sub:Nn} \langle \texttt{muskip} \rangle \{\langle \texttt{muskip expression} \rangle \}
\end{verbatim}

Subtracts the result of the \langle \texttt{muskip expression} \rangle from the current content of the \langle \texttt{muskip} \rangle.

20 Using \texttt{muskip} expressions and variables

\begin{verbatim}
\texttt{\textbackslash muskip\_eval:n} \* \texttt{\textbackslash muskip\_eval:n} \{\langle \texttt{muskip expression} \rangle \}
\end{verbatim}

Evaluates the \langle \texttt{muskip expression} \rangle, expanding any skips and token list variables within the \langle \texttt{expression} \rangle to their content (without requiring \texttt{\textbackslash muskip\_use:N} / \texttt{\tl\_use:N}) and applying the standard mathematical rules. The result of the calculation is left in the input stream as a \langle \texttt{muglue\_denotation} \rangle after two expansions. This will be expressed in \texttt{mu}, and will require suitable termination if used in a \texttt{TeX}-style assignment as it is \textit{not} an \langle \texttt{internal\_muglue} \rangle.

\begin{verbatim}
\texttt{\textbackslash muskip\_use:N} \* \texttt{\textbackslash muskip\_use:n} \texttt{\textbackslash muskip\_use:c} \*
\end{verbatim}

Recovers the content of a \langle \texttt{skip} \rangle and places it directly in the input stream. An error will be raised if the variable does not exist or if it is invalid. Can be omitted in places where a \langle \texttt{dimension} \rangle is required (such as in the argument of \texttt{\textbackslash muskip\_eval:n}).

\texttt{\TeX\_hackers note:} \texttt{\textbackslash muskip\_use:N} is the \TeX\ primitive \texttt{\textbackslash the}: this is one of several \LaTeX3\ names for this primitive.
21 Viewing muskip variables

\texttt{\texttt{\textbackslash mskip\_show\_N}}\texttt{\textbackslash mskip\_show\_C}  
\texttt{\textbackslash mskip\_show\_N \langle mskip \rangle}  
\texttt{\textbackslash mskip\_show\_C}  
Displays the value of the \texttt{\langle mskip \rangle} on the terminal.

\texttt{\textbackslash mskip\_show\_N}  
\texttt{\textbackslash mskip\_show\_N \{\textit{mskip expression}\}}  
Displays the result of evaluating the \texttt{\langle mskip expression \rangle} on the terminal.

New: 2011-11-22  
Updated: 2012-05-27

22 Constant muskips

\texttt{\textbackslash c\_max\_mskip}  
The maximum value that can be stored as a muskip, with no stretch nor shrink component.

\texttt{\textbackslash c\_zero\_mskip}  
A zero length as a muskip, with no stretch nor shrink component.

23 Scratch muskips

\texttt{\textbackslash l\_tmpa\_mskip}  
\texttt{\textbackslash l\_tmpb\_mskip}  
Scratch muskip for local assignment. These are never used by the kernel code, and so are safe for use with any \LaTeX{}3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

\texttt{\textbackslash g\_tmpa\_mskip}  
\texttt{\textbackslash g\_tmpb\_mskip}  
Scratch muskip for global assignment. These are never used by the kernel code, and so are safe for use with any \LaTeX{}3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

24 Primitive conditional

\texttt{\textbackslash if\_dim:w}  
\texttt{\textbackslash if\_dim:w \{dimen\_1\} \{relation\} \{dimen\_2\}}  
\texttt{(true code)}  
\texttt{\textbackslash else:}  
\texttt{\{false\}}  
\texttt{\textbackslash fi:}  
Compare two dimensions. The \texttt{\langle relation \rangle} is one of \texttt{<}, \texttt{=} or \texttt{>} with category code 12.

\TeX{}hackers note: This is the \TeX{} primitive \texttt{\textbackslash ifdim}. 

90
25 Internal functions

\__dim_eval:w \* \__dim_eval:w ⟨dimexpr⟩ \__dim_eval_end:
\__dim_eval_end: \*

Evaluates ⟨dimension expression⟩ as described for \dim_eval:n. The evaluation stops when an unexpandable token which is not a valid part of a dimension is read or when \__dim_eval_end: is reached. The latter is gobbled by the scanner mechanism: \__dim_eval_end: itself is unexpandable but used correctly the entire construct is expandable.

\TeXhackers note: This is the \vTeX primitive \dimexpr.
Part XI
The \l3tl package
Token lists

\TeX\ works with tokens, and \LaTeX3\ therefore provides a number of functions to deal with
lists of tokens. Token lists may be present directly in the argument to a function:

\begin{verbatim}
\texttt{\textbackslash foo:n \{ a collection of \texttt{\textbackslash tokens} \}}
\end{verbatim}

or may be stored in a so-called “token list variable”, which have the suffix \texttt{tl}: a token
list variable can also be used as the argument to a function, for example

\begin{verbatim}
\texttt{\textbackslash foo:N \l\_some\_tl}
\end{verbatim}

In both cases, functions are available to test and manipulate the lists of tokens, and these
have the module prefix \texttt{tl}. In many cases, function which can be applied to token list
variables are paired with similar functions for application to explicit lists of tokens: the
two “views” of a token list are therefore collected together here.

A token list (explicit, or stored in a variable) can be seen either as a list of “items”,
or a list of “tokens”. An item is whatever \texttt{\use:n} would grab as its argument: a single
non-space token or a brace group, with optional leading explicit space characters (each
item is thus itself a token list). A token is either a normal \texttt{N} argument, or \texttt{|, \{, or \}}
(assuming normal \TeX\ category codes). Thus for example

\begin{verbatim}
\{ Hello \} \texttt{\textbackslash \textbackslash world}
\end{verbatim}

contains six items (Hello, w, o, r, l and d), but thirteen tokens (\{, H, e, l, l, o, \}, \|, w,
o, r, l and d). Functions which act on items are often faster than their analogue acting
directly on tokens.

\textbf{\TeX\ hackers note:} When \TeX\ fetches an undelimited argument from the input stream,
explicit character tokens with character code 32 (space) and category code 10 (space), which we
here call “explicit space characters”, are ignored. If the following token is an explicit character
token with category code 1 (begin-group) and an arbitrary character code, then \TeX\ scans ahead
to obtain an equal number of explicit character tokens with category code 1 (begin-group) and 2
(end-group), and the resulting list of tokens (with outer braces removed) becomes the argument.
Otherwise, a single token is taken as the argument for the macro: we call such single tokens
“N-type”, as they are suitable to be used as an argument for a function with the signature \texttt{:N}.

When \TeX\ reads a character of category code 10 for the first time, it is converted to an
explicit space character, with character code 32, regardless of the initial character code. “Funny”
spaces with a different category code, can be produced using \texttt{\textbackslash tl\_to\_lowercase:n} or \texttt{\textbackslash tl\_to\_\-\_uppercase:n}. Explicit space characters are also produced as a result of \texttt{\textbackslash token\_to\_str:N}, \texttt{\textbackslash tl\_to\_str:n}, etc.
1 Creating and initialising token list variables

\tl_new:N
\tl_new:c

Creates a new \langle tl var \rangle or raises an error if the name is already taken. The declaration is global. The \langle tl var \rangle will initially be empty.

\tl_const:Nn
\tl_const:(N|N|c|c)

Creates a new constant \langle tl var \rangle or raises an error if the name is already taken. The value of the \langle tl var \rangle will be set globally to the \langle token list \rangle.

\tl_clear:N
\tl_clear:c
\tl_gclear:N
\tl_gclear:c

Clears all entries from the \langle tl var \rangle.

\tl_clear_new:N
\tl_clear_new:c
\tl_gclear_new:N
\tl_gclear_new:c

Ensures that the \langle tl var \rangle exists globally by applying \tl_new:N if necessary, then applies \tl_\(g\)clear:N to leave the \langle tl var \rangle empty.

\tl_set_eq:NN
\tl_set_eq:(cN|Nc|cc)
\tl_gset_eq:NN
\tl_gset_eq:(cN|Nc|cc)

Sets the content of \langle tl var_1 \rangle equal to that of \langle tl var_2 \rangle.

\tl_concat:NNN
\tl_concat:ccc
\tl_gconcat:NNN
\tl_gconcat:ccc

Concatenates the content of \langle tl var_1 \rangle and \langle tl var_2 \rangle together and saves the result in \langle tl var_3 \rangle. The \langle tl var_2 \rangle will be placed at the left side of the new token list.

\tl_if_exist_p:N
\tl_if_exist_p:c
\tl_if_exist:NTF
\tl_if_exist:cTF

Tests whether the \langle tl var \rangle is currently defined. This does not check that the \langle tl var \rangle really is a token list variable.
2 Adding data to token list variables

\tl_set:Nn \tl_set:Nn \tl_set:Nn \tl_set:Nn
⟨tl var⟩ \langle tokens⟩ \langle tokens⟩ \langle tokens⟩
\tl_gset:Nn \tl_gset:Nn \tl_gset:Nn \tl_gset:Nn
⟨tl var⟩ \langle tokens⟩ \langle tokens⟩ \langle tokens⟩

Sets \langle tl var \rangle to contain \langle tokens \rangle, removing any previous content from the variable.

\tl_put_left:Nn \tl_put_left:Nn \tl_put_left:Nn \tl_put_left:Nn
⟨tl var⟩ \langle tokens⟩ \langle tokens⟩ \langle tokens⟩
\tl_gput_left:Nn \tl_gput_left:Nn \tl_gput_left:Nn \tl_gput_left:Nn
⟨tl var⟩ \langle tokens⟩ \langle tokens⟩ \langle tokens⟩

Appends \langle tokens \rangle to the left side of the current content of \langle tl var \rangle.

\tl_put_right:Nn \tl_put_right:Nn \tl_put_right:Nn \tl_put_right:Nn
⟨tl var⟩ \langle tokens⟩ \langle tokens⟩ \langle tokens⟩
\tl_gput_right:Nn \tl_gput_right:Nn \tl_gput_right:Nn \tl_gput_right:Nn
⟨tl var⟩ \langle tokens⟩ \langle tokens⟩ \langle tokens⟩

Appends \langle tokens \rangle to the right side of the current content of \langle tl var \rangle.

3 Modifying token list variables

\tl_replace_once:Nnn \tl_replace_once:Nnn \tl_replace_once:Nnn \tl_replace_once:cn
\tl_greplace_once:Nnn \tl_greplace_once:Nnn \tl_greplace_once:cn
\tl replace_once:Nn \tl replace_once:cn \tl_greplace_once:Nn \tl_greplace_once:cn
\tl_replace_all:Nnn \tl_replace_all:Nnn \tl_replace_all:Nnn \tl_replace_all:cn
\tl_greplace_all:Nnn \tl_greplace_all:Nnn \tl_greplace_all:cn
\tl replace_all:Nn \tl replace_all:cn \tl_greplace_all:Nn \tl_greplace_all:cn
\tl_remove_once:Nn \tl_remove_once:cn \tl_gremove_once:Nn \tl_gremove_once:cn
Updated: 2011-08-11

\tl_replace_once:Nnn \tl_replace_once:Nnn \tl_replace_once:Nnn \tl_replace_once:cn
\tl_greplace_once:Nnn \tl_greplace_once:Nnn \tl_greplace_once:cn
\tl replace_once:Nn \tl replace_once:cn \tl_greplace_once:Nn \tl_greplace_once:cn
\tl_replace_all:Nnn \tl_replace_all:Nnn \tl_replace_all:Nnn \tl_replace_all:cn
\tl_greplace_all:Nnn \tl_greplace_all:Nnn \tl_greplace_all:cn
\tl replace_all:Nn \tl replace_all:cn \tl_greplace_all:Nn \tl_greplace_all:cn
\tl_remove_once:Nn \tl_remove_once:cn \tl_gremove_once:Nn \tl_gremove_once:cn
Updated: 2011-08-11

\tl_replace_once:Nnn \tl_replace_once:cn
\tl_greplace_once:Nnn \tl_greplace_once:cn
\tl replace_once:Nn \tl replace_once:cn
\tl_greplace_all:Nnn \tl_greplace_all:cn
\tl replace_all:Nn \tl replace_all:cn
\tl_gremove_once:Nn \tl_gremove_once:cn
Updated: 2011-08-11

\tl_replace_once:Nnn \tl_replace_once:cn
\tl_greplace_once:Nnn \tl_greplace_once:cn

Replaces the first (leftmost) occurrence of \langle old tokens \rangle in the \langle tl var \rangle with \langle new tokens \rangle. \langle Old tokens \rangle cannot contain {, } or # (more precisely, explicit character tokens with category code 1 (begin-group) or 2 (end-group), and tokens with category code 6).

\tl_replace_all:Nnn \tl_replace_all:cn
\tl_greplace_all:Nnn \tl_greplace_all:cn
\tl replace_all:Nn \tl replace_all:cn
\tl_greplace_all:Nn \tl_greplace_all:cn

Replaces all occurrences of \langle old tokens \rangle in the \langle tl var \rangle with \langle new tokens \rangle. \langle Old tokens \rangle cannot contain {, } or # (more precisely, explicit character tokens with category code 1 (begin-group) or 2 (end-group), and tokens with category code 6). As this function operates from left to right, the pattern \langle old tokens \rangle may remain after the replacement (see \tl_remove_all:Nn for an example).

\tl_remove_once:Nn \tl_remove_once:cn
\tl_gremove_once:Nn \tl_gremove_once:cn
\tl remove_once:Nn \tl remove_once:cn
\tl_gremove_all:Nn \tl_gremove_all:cn
\tl remove_all:Nn \tl remove_all:cn

Removes the first (leftmost) occurrence of \langle tokens \rangle from the \langle tl var \rangle. \langle Tokens \rangle cannot contain {, } or # (more precisely, explicit character tokens with category code 1 (begin-group) or 2 (end-group), and tokens with category code 6).
\tl_remove_all:Nn \tl_remove_all:cn \tl_gremove_all:Nn \tl_gremove_all:cn
Updated: 2011-08-11

Removes all occurrences of \textit{(tokens)} from the \textit{(tl var)}. \textit{(Tokens)} cannot contain \{, \} or # (more precisely, explicit character tokens with category code 1 (begin-group) or 2 (end-group), and tokens with category code 6). As this function operates from left to right, the pattern \textit{(tokens)} may remain after the removal, for instance,

\texttt{\textbackslash l_set:Nn \l_tmpa_tl \{abbccd\} \textbackslash l_remove_all:Nn \l_tmpa_tl \{bc\}}

will result in \texttt{\l_tmpa_tl} containing \texttt{abcd}.

### 4 Reassigning token list category codes

\tl_set_rescan:Nnn \tl_set_rescan:(Nno|Nnx|cnn|cno|cnx) \tl_gset_rescan:Nnn \tl_gset_rescan:(Nno|Nnx|cnn|cno|cnx)
Updated: 2011-12-18

Sets \textit{(tl var)} to contain \textit{(tokens)}, applying the category code régime specified in the \textit{(setup)} before carrying out the assignment. This allows the \textit{(tl var)} to contain material with category codes other than those that apply when \textit{(tokens)} are absorbed. Trailing spaces at the end of the \textit{(tokens)} are discarded in the rescanning process. The \textit{(setup)} is not limited to changes of category code but may contain any valid input, for example assignment of the expansion of active tokens. See also \texttt{\textbackslash tl_rescan:nn}.

\tl_rescan:nn
Updated: 2011-12-18

Recans \textit{(tokens)} applying the category code régime specified in the \textit{(setup)}, and leaves the resulting tokens in the input stream. Trailing spaces at the end of the \textit{(tokens)} are discarded in the rescanning process. The \textit{(setup)} is not limited to changes of category code but may contain any valid input, for example assignment of the expansion of active tokens. See also \texttt{\textbackslash tl_set_rescan:Nnn}.

### 5 Reassigning token list character codes

\tl_to_lowercase:n
Updated: 2012-09-08

Works through all of the \textit{(tokens)}, replacing each character token with the lower case equivalent as defined by \texttt{\textbackslash char_set_lccode:nn}. Characters with no defined lower case character code are left unchanged. This process does not alter the category code assigned to the \textit{(tokens)}.

\textbf{\LaTeX{}hackers note:} This is a wrapper around the \LaTeX{} primitive \texttt{\lowercase}. 

95
\tl_to_uppercase:n \{\langle \text{tokens} \rangle \} \n
Works through all of the \langle \text{tokens} \rangle, replacing each character token with the upper case equivalent as defined by \char_set_uccode:nn. Characters with no defined upper case character code are left unchanged. This process does not alter the category code assigned to the \langle \text{tokens} \rangle.

\textbf{\LaTeX}hackers note: This is a wrapper around the \LaTeX primitive \texttt{\uppercase}.  

6 Token list conditionals

\tl_if_blank_p:n \{\langle \text{token list} \rangle \} \n
\tl_if_blank:p:\langle \text{token list} \rangle \n
\tl_if_blank:nTF \{\langle \text{token list} \rangle \} \{\langle \text{true code} \rangle \} \{\langle \text{false code} \rangle \} \n
Tests if the \langle \text{token list} \rangle consists only of blank spaces (\textit{i.e.} contains no item). The test is \texttt{true} if \langle \text{token list} \rangle is zero or more explicit space characters (explicit tokens with character code 32 and category code 10), and is \texttt{false} otherwise.

\tl_if_empty_p:N \langle \text{tl var} \rangle \n
\tl_if_empty:p:c \langle \text{tl var} \rangle \n
\tl_if_empty:nTF \langle \text{tl var} \rangle \{\langle \text{true code} \rangle \} \{\langle \text{false code} \rangle \} \n
Tests if the \langle \text{token list variable} \rangle is entirely empty (\textit{i.e.} contains no tokens at all).

\tl_if_eq_p:NN \langle \text{tl var}_1 \rangle \{\langle \text{tl var}_2 \rangle \} \n
\tl_if_eq:NNTF \langle \text{tl var}_1 \rangle \{\langle \text{tl var}_2 \rangle \} \{\langle \text{true code} \rangle \} \{\langle \text{false code} \rangle \} \n
Compares the content of two \langle \text{token list variables} \rangle and is logically \texttt{true} if the two contain the same list of tokens (\textit{i.e.} identical in both the list of characters they contain and the category codes of those characters). Thus for example

\begin{verbatim}
\tl_set:Nn \l_tmpa_tl { abc }
\tl_set:Nx \l_tmpb_tl { \tl_to_str:n \{ abc \} }
\tl_if_eq:NNTF \l_tmpa_tl \l_tmpb_tl \{ true \} \{ false \}
\end{verbatim}

yields \texttt{false}.

\tl_if_eq:nnTF \langle \text{token list}_1 \rangle \{\langle \text{token list}_2 \rangle \} \{\langle \text{true code} \rangle \} \{\langle \text{false code} \rangle \} \n
Tests if \langle \text{token list}_1 \rangle and \langle \text{token list}_2 \rangle contain the same list of tokens, both in respect of character codes and category codes.
\tl_if_in:NnTF \tl_if_in:cnTF
\tl_if_in:nTF \tl_if_in:(Vn|on|no)TF

Tests if the \langle token list \rangle is found in the content of the \langle tl var \rangle. The \langle token list \rangle cannot contain the tokens \{, \} or \# (more precisely, explicit character tokens with category code 1 (begin-group) or 2 (end-group), and tokens with category code 6).

\tl_if_single_p:N \tl_if_single:NTF \tl_if_single:cTF
\tl_if_single_p:n \tl_if_single:nTF \tl_if_single:cTF

Tests if the content of the \langle tl var \rangle consists of a single item, \textit{i.e.} is a single normal token (neither an explicit space character nor a begin-group character) or a single brace group, surrounded by optional spaces on both sides. In other words, such a token list has token count 1 according to \tl_count:N.

\tl_case:NnTF \tl_case:cnTF

New: 2013-07-24

This function compares the \langle test token list variable \rangle in turn with each of the \langle token list variable cases \rangle. If the two are equal (as described for \tl_if_eq:NNTF) then the associated \langle code \rangle is left in the input stream. If any of the cases are matched, the \langle true code \rangle is also inserted into the input stream (after the code for the appropriate case), while if none match then the \langle false code \rangle is inserted. The function \tl_case:Nn, which does nothing if there is no match, is also available.
7 Mapping to token lists

\texttt{tl\_map\_function:NN} \hspace{1em}$\star$
\texttt{tl\_map\_function:cN} \hspace{1em}$\star$

\texttt{tl\_map\_function:NN} \{tl var\} \{function\}

Applies \{function\} to every \{item\} in the \{tl var\}. The \{function\} will receive one argument for each iteration. This may be a number of tokens if the \{item\} was stored within braces. Hence the \{function\} should anticipate receiving n-type arguments. See also \texttt{tl\_map\_function:nN}.

\texttt{tl\_map\_function:nN} \hspace{1em}$\star$

\texttt{tl\_map\_function:nN} \{token list\} \{function\}

Applies \{function\} to every \{item\} in the \{token list\}. The \{function\} will receive one argument for each iteration. This may be a number of tokens if the \{item\} was stored within braces. Hence the \{function\} should anticipate receiving n-type arguments. See also \texttt{tl\_map\_function:nN}.

\texttt{tl\_map\_inline:Nn}
\texttt{tl\_map\_inline:cn}

\texttt{tl\_map\_inline:Nn} \{tl var\} \{(inline function)\}

Applies the \{inline function\} to every \{item\} stored within the \{tl var\}. The \{inline function\} should consist of code which will receive the \{item\} as #1. One in line mapping can be nested inside another. See also \texttt{tl\_map\_function:NN}.

\texttt{tl\_map\_inline:nn}

\texttt{tl\_map\_inline:nn} \{token list\} \{(inline function)\}

Applies the \{inline function\} to every \{item\} stored within the \{token list\}. The \{inline function\} should consist of code which will receive the \{item\} as #1. One in line mapping can be nested inside another. See also \texttt{tl\_map\_function:nN}.

\texttt{tl\_map\_variable:NNn}
\texttt{tl\_map\_variable:cNn}

\texttt{tl\_map\_variable:NNn} \{tl var\} \{variable\} \{(function)\}

Applies the \{function\} to every \{item\} stored within the \{tl var\}. The \{function\} should consist of code which will receive the \{item\} stored in the \{variable\}. One variable mapping can be nested inside another. See also \texttt{tl\_map\_inline:Nn}.

\texttt{tl\_map\_variable:nNn}

\texttt{tl\_map\_variable:nNn} \{token list\} \{variable\} \{(function)\}

Applies the \{function\} to every \{item\} stored within the \{token list\}. The \{function\} should consist of code which will receive the \{item\} stored in the \{variable\}. One variable mapping can be nested inside another. See also \texttt{tl\_map\_inline:nn}.
\tl_map_break: \tl_map_break:

Used to terminate a \tl_map_\ldots function before all entries in the ⟨token list variable⟩ have been processed. This will normally take place within a conditional statement, for example

\tl_map_inline:Nn \l_my_tl
{\str_if_eq:nnT { #1 } { bingo } { \tl_map_break: }
% Do something useful
}

See also \tl_map_break:n. Use outside of a \tl_map_\ldots scenario will lead to low level \TeX\ errors.

\TeXhackers\note: When the mapping is broken, additional tokens may be inserted by the internal macro \__prg_break_point:Nn before the ⟨tokens⟩ are inserted into the input stream. This will depend on the design of the mapping function.

\tl_map_break:n \tl_map_break:n ⟨tokens⟩

Used to terminate a \tl_map_\ldots function before all entries in the ⟨token list variable⟩ have been processed, inserting the ⟨tokens⟩ after the mapping has ended. This will normally take place within a conditional statement, for example

\tl_map_inline:Nn \l_my_tl
{\str_if_eq:nnT { #1 } { bingo }
 { \tl_map_break:n { ⟨tokens⟩ } }
% Do something useful
}

Use outside of a \tl_map_\ldots scenario will lead to low level \TeX\ errors.

\TeXhackers\note: When the mapping is broken, additional tokens may be inserted by the internal macro \__prg_break_point:Nn before the ⟨tokens⟩ are inserted into the input stream. This will depend on the design of the mapping function.
8 Using token lists

\tl_to_str:n \{\langle\text{token list}\rangle\}

Converts the \langle\text{token list}\rangle to a \langle\text{string}\rangle, leaving the resulting character tokens in the input stream. A \langle\text{string}\rangle is a series of tokens with category code 12 (other) with the exception of spaces, which retain category code 10 (space).

\textbf{\LaTeX} hackers note: Converting a \langle\text{token list}\rangle to a \langle\text{string}\rangle yields a concatenation of the string representations of every token in the \langle\text{token list}\rangle. The string representation of a control sequence is

- an escape character, whose character code is given by the internal parameter \texttt{\textbackslash escapechar}, absent if the \texttt{\textbackslash escapechar} is negative or greater than the largest character code;
- the control sequence name, as defined by \texttt{\cs_to_str:N};
- a space, unless the control sequence name is a single character whose category at the time of expansion of \texttt{\tl_to_str:n} is not “letter”.

The string representation of an explicit character token is that character, doubled in the case of (explicit) macro parameter characters (normally \#). In particular, the string representation of a token list may depend on the category codes in effect when it is evaluated, and the value of the \texttt{\textbackslash escapechar}: for instance \texttt{\tl_to_str:n \{\textbackslash a\}} normally produces the three character “backslash”, “lower-case a”, “space”, but it may also produce a single “lower-case a” if the escape character is negative and \texttt{a} is currently not a letter.

\tl_to_str:N \texttt{\langle tl var\rangle}
\tl_to_str:c

Converts the content of the \langle\text{tl var}\rangle into a series of characters with category code 12 (other) with the exception of spaces, which retain category code 10 (space). This \langle\text{string}\rangle is then left in the input stream. For low-level details, see the notes given for \texttt{\tl_to_str:n}.

\tl_use:N \texttt{\langle tl var\rangle}
\tl_use:c

Recover the content of a \langle\text{tl var}\rangle and places it directly in the input stream. An error will be raised if the variable does not exist or if it is invalid. Note that it is possible to use a \langle\text{tl var}\rangle directly without an accessor function.

9 Working with the content of token lists

\tl_count:n \texttt{\langle\text{tokens}\rangle}
\tl_count:V \texttt{\langle\text{tokens}\rangle}
\tl_count:o \texttt{\langle\text{tokens}\rangle}

Counts the number of \langle\text{items}\rangle in \langle\text{tokens}\rangle and leaves this information in the input stream. Unbraced tokens count as one element as do each token group \langle\ldots\rangle. This process will ignore any unprotected spaces within \langle\text{tokens}\rangle. See also \texttt{\tl_count:N}. This function requires three expansions, giving an \langle\text{integer denotation}\rangle.
\texttt{\textbackslash tl\_count:N} \texttt{\textbackslash tl\_count:c} \texttt{\textbackslash new: 2012-05-13}

Counts the number of token groups in the \langle \texttt{tl var} \rangle and leaves this information in the input stream. Unbraced tokens count as one element as do each token group \{ \ldots \}. This process will ignore any unprotected spaces within the \langle \texttt{tl var} \rangle. See also \texttt{\textbackslash tl\_count:n}. This function requires three expansions, giving an \langle integer denotation \rangle.

\texttt{\textbackslash tl\_reverse:n} \texttt{\textbackslash tl\_reverse:\langle V|o \rangle} \texttt{\textbackslash updated: 2012-01-08}

Reverses the order of the \langle items \rangle in the \langle token list \rangle, so that \langle item_1 \rangle \langle item_2 \rangle \langle item_3 \rangle \ldots \langle item_n \rangle becomes \langle item_n \rangle \ldots \langle item_3 \rangle \langle item_2 \rangle \langle item_1 \rangle. This process will preserve unprotected space within the \langle token list \rangle. Tokens are not reversed within braced token groups, which keep their outer set of braces. In situations where performance is important, consider \texttt{\textbackslash tl\_reverse\_items:n}. See also \texttt{\textbackslash tl\_reverse:N}.

\texttt{\textbackslash tl\_reverse\_items:n} \texttt{\textbackslash new: 2012-01-08}

Reverses the order of the \langle items \rangle stored in \langle tl var \rangle, so that \langle item_1 \rangle \langle item_2 \rangle \langle item_3 \rangle \ldots \langle item_n \rangle becomes \langle item_n \rangle \ldots \langle item_3 \rangle \langle item_2 \rangle \langle item_1 \rangle. This process will preserve unprotected spaces within the \langle token list variable \rangle. Braced token groups are copied without reversing the order of tokens, but keep the outer set of braces. Items which are initially not braced are copied with braces in the result. In cases where preserving spaces is important, consider the slower function \texttt{\textbackslash tl\_reverse:n}. See also \texttt{\textbackslash tl\_reverse\_items:n}.

\texttt{\textbackslash tl\_trim\_spaces:n} \texttt{\textbackslash new: 2011-07-09} \texttt{\textbackslash updated: 2012-06-25}

Removes any leading and trailing explicit space characters (explicit tokens with character code 32 and category code 10) from the \langle token list \rangle and leaves the result in the input stream.

\texttt{\textbackslash tl\_trim\_spaces:n} \texttt{\textbackslash new: 2011-07-09} \texttt{\textbackslash updated: 2012-06-25}

The result is returned within \texttt{\textbackslash unexpanded}, which means that the token list will not expand further when appearing in an \texttt{x}\textbackslash type argument expansion.
\texttt{\tl_trim_spaces:N} \texttt{(tl var)}

Removes any leading and trailing explicit space characters (explicit tokens with character code 32 and category code 10) from the content of the \texttt{tl var}. Note that this therefore \textit{resets} the content of the variable.

10 The first token from a token list

Functions which deal with either only the very first item (balanced text or single normal token) in a token list, or the remaining tokens.

\texttt{\tl_head:n \{ (token list) \}}

Leaves in the input stream the first \texttt{item} in the \texttt{(token list)}, discarding the rest of the \texttt{(token list)}. All leading explicit space characters (explicit tokens with character code 32 and category code 10) are discarded; for example

\begin{verbatim}
\tl_head:n \{ abc \}
\end{verbatim}

and

\begin{verbatim}
\tl_head:n \{ - abc \}
\end{verbatim}

will both leave \texttt{a} in the input stream. If the “head” is a brace group, rather than a single token, the braces will be removed, and so

\begin{verbatim}
\tl_head:n \{ - \{ - ab \} c \}
\end{verbatim}

yields \texttt{ab}. A blank \texttt{(token list)} (see \texttt{\tl_if_blank:nF}) will result in \texttt{\tl_head:n} leaving nothing in the input stream.

\TeXhacksnote{The result is returned within \texttt{\exp_not:n}, which means that the token list will not expand further when appearing in an \texttt{x}-type argument expansion.}

\texttt{\tl_head:w \{ (token list) \} \{ \}} \texttt{\q_stop}

Leaves in the input stream the first \texttt{item} in the \texttt{(token list)}, discarding the rest of the \texttt{(token list)}. All leading explicit space characters (explicit tokens with character code 32 and category code 10) are discarded. A blank \texttt{(token list)} (which consists only of space characters) will result in a low-level \TeX{} error, which may be avoided by the inclusion of an empty group in the input (as shown), without the need for an explicit test. Alternatively, \texttt{\tl_if_blank:nF} may be used to avoid using the function with a “blank” argument. This function requires only a single expansion, and thus is suitable for use within an \texttt{o}-type expansion. In general, \texttt{\tl_head:n} should be preferred if the number of expansions is not critical.
Discards all leading explicit space characters (explicit tokens with character code 32 and category code 10) and the first ⟨item⟩ in the ⟨token list⟩, and leaves the remaining tokens in the input stream. Thus for example

\tl_tail:n { a - {bc} d }

and

\tl_tail:n { - a - {bc} d }

will both leave ⟨bc⟩d in the input stream. A blank ⟨token list⟩ (see \tl_if_blank:nTF) will result in \tl_tail:n leaving nothing in the input stream.

\TeX{hackers note}: The result is returned within \exp_not:n, which means that the token list will not expand further when appearing in an \x-type argument expansion.

\tl_if_head_eq_catcode_p:nN \tl_if_head_eq_catcode:nN
\tl_if_head_eq_meaning_p:nN \tl_if_head_eq_meaning:nN
\tl_if_head_is_group_p:n \tl_if_head_is_group:n

Tests if the first ⟨token⟩ in the ⟨token list⟩ has the same category code as the ⟨test token⟩. In the case where the ⟨token list⟩ is empty, the test will always be false.

Tests if the first ⟨token⟩ in the ⟨token list⟩ has the same character code as the ⟨test token⟩. In the case where the ⟨token list⟩ is empty, the test will always be false.

Tests if the first ⟨token⟩ in the ⟨token list⟩ has the same meaning as the ⟨test token⟩. In the case where ⟨token list⟩ is empty, the test will always be false.

Tests if the first ⟨token⟩ in the ⟨token list⟩ is an explicit begin-group character (with category code 1 and any character code), in other words, if the ⟨token list⟩ starts with a brace group. In particular, the test is false if the ⟨token list⟩ starts with an implicit token such as \c_group_begin_token, or if it is empty. This function is useful to implement actions on token lists on a token by token basis.
\tl_if_head_is_N_type_p:n \{\text{token list}\} \tl_if_head_is_N_type:nTF \{\text{token list}\} \{\text{true code}\} \{\text{false code}\}

Tests if the first \langle token \rangle in the \langle token list \rangle is a normal N-type argument. In other words, it is neither an explicit space character (explicit token with character code 32 and category code 10) nor an explicit begin-group character (with category code 1 and any character code). An empty argument yields \text{false}, as it does not have a “normal” first token. This function is useful to implement actions on token lists on a token by token basis.

\tl_if_head_is_space_p:n \{\text{token list}\} \tl_if_head_is_space:nTF \{\text{token list}\} \{\text{true code}\} \{\text{false code}\}

Tests if the first \langle token \rangle in the \langle token list \rangle is an explicit space character (explicit token with character code 12 and category code 10). In particular, the test is \text{false} if the \langle token list \rangle starts with an implicit token such as \c_space_token, or if it is empty. This function is useful to implement actions on token lists on a token by token basis.

11 Using a single item

\tl_item:nn \{\text{token list}\} \{\text{integer expression}\} \tl_item:Nn \tl_item:cn

Indexing items in the \langle token list \rangle from 1 on the left, this function will evaluate the \langle integer expression \rangle and leave the appropriate item from the \langle token list \rangle in the input stream. If the \langle integer expression \rangle is negative, indexing occurs from the right of the token list, starting at \text{−1} for the right-most item. If the index is out of bounds, then the function expands to nothing.

\TeXhackersnote: The result is returned within the \texttt{unexpanded} primitive (\exp_not:n), which means that the \langle item \rangle will not expand further when appearing in an \texttt{x}-type argument expansion.

12 Viewing token lists

\tl_show:N \tl_show:c

Displays the content of the \langle tl var \rangle on the terminal.

\TeXhackersnote: This is similar to the \TeX primitive \show, wrapped to a fixed number of characters per line.
\tl_show:n \tl_show:n \langle \text{token list} \rangle 
Displays the \langle \text{token list} \rangle on the terminal.

\textbf{\LaTeX}hackers note: This is similar to the \LaTeX primitive \texttt{\showtokens}, wrapped to a fixed number of characters per line.

13 Constant token lists

\c_empty_tl
Constant that is always empty.

\c_job_name_tl
Constant that gets the “job name” assigned when \LaTeX starts.

\textbf{\LaTeX}hackers note: This copies the contents of the primitive \texttt{\jobname}. It is a constant that is set by \LaTeX and should not be overwritten by the package.

\c_space_tl
An explicit space character contained in a token list (compare this with \c_space_token). For use where an explicit space is required.

14 Scratch token lists

\l_tmpa_tl
\l_tmpb_tl
Scratch token lists for local assignment. These are never used by the kernel code, and so are safe for use with any \LaTeX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

\g_tmpa_tl
\g_tmpb_tl
Scratch token lists for global assignment. These are never used by the kernel code, and so are safe for use with any \LaTeX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

15 Internal functions

\_\_tl_trim_spaces:nn \_\_tl_trim_spaces:nn \{ \q_mark \langle \text{token list} \rangle \} \langle \text{continuation} \rangle
This function removes all leading and trailing explicit space characters from the \langle \text{token list} \rangle, and expands to the \langle \text{continuation} \rangle, followed by a brace group containing \texttt{use-none:n} \q_mark (trimmed token list). For instance, \texttt{\tl_trim_spaces:n} is implemented by taking the \langle \text{continuation} \rangle to be \texttt{\exp_not:o}, and the \texttt{o}-type expansion removes the \q_mark. This function is also used in \texttt{l3clist} and \texttt{l3candidates}.
Part XII
The \texttt{l3str} package
Strings

\LaTeX{} associates each character with a category code: as such, there is no concept of a “string” as commonly understood in many other programming languages. However, there are places where we wish to manipulate token lists while in some sense “ignoring” category codes: this is done by treating token lists as strings in a \texttt{TeX} sense.

A \texttt{TeX} string (and thus an \texttt{expl3} string) is a series of characters which have category code 12 (“other”) with the exception of space characters which have category code 10 (“space”). Thus at a technical level, a \texttt{TeX} string is a token list with the appropriate category codes. In this documentation, these will simply be referred to as strings: note that they can be stored in token lists as normal.

The functions documented here take literal token lists, convert to strings and then carry out manipulations. Thus they may informally be described as “ignoring” category code. Note that the functions \texttt{\cs_to_str:N}, \texttt{\tl_to_str:n}, \texttt{\tl_to_str:N} and \texttt{\token_to_str:N} (and variants) will generate strings from the appropriate input: these are documented in \texttt{l3basics}, \texttt{l3tl} and \texttt{l3token}, respectively.

\section{The first character from a string}

\begin{itemize}
  \item \texttt{\str_head:n \{token list\}}
  \item \texttt{\str_tail:n \{token list\}}
\end{itemize}

Rev: 2011-08-10

Converts the (token list) into a string, as described for \texttt{\tl_to_str:n}. The \texttt{\str_head:n} function then leaves the first character of this string in the input stream. The \texttt{\str_tail:n} function leaves all characters except the first in the input stream. The first character may be a space. If the (token list) argument is entirely empty, nothing is left in the input stream.

\subsection{Tests on strings}

\begin{itemize}
  \item \texttt{\str_if_eq_p:nn \{\{tl\}_1\} \{\{tl\}_2\}}
  \item \texttt{\str_if_eq:(\{Vn\}|\{on\}|\{no\}|\{nV\}|\{VV\}) \{\{true code\}\} \{\{false code\}\}}
  \item \texttt{\str_if_eq_p:nnTF \{\{tl\}_1\} \{\{tl\}_2\}}
  \item \texttt{\str_if_eq:(\{Vn\}|\{on\}|\{no\}|\{nV\}|\{VV\})TF}
\end{itemize}

Compares the two (token lists) on a character by character basis, and is \texttt{true} if the two lists contain the same characters in the same order. Thus for example

\begin{verbatim}
\str_if_eq_p:no \{ abc \} \{ \tl_to_str:n \{ abc \} \}
\end{verbatim}

is logically \texttt{true}.
\str_if_eq_x_p:nn \str_if_eq_x_p:nn \str_if_eq_x_p:nn \str_if_eq_x_p:nn \str_if_eq_x_p:nn \str_if_eq_x_p:nn \str_if_eq_x_p:nn

\str_if_eq_x:nnTF \str_if_eq_x:nnTF \str_if_eq_x:nnTF \str_if_eq_x:nnTF \str_if_eq_x:nnTF \str_if_eq_x:nnTF \str_if_eq_x:nnTF

Compares the full expansion of two \textit{token lists} on a character by character basis, and is \textit{true} if the two lists contain the same characters in the same order. Thus for example

\begin{verbatim}
\str_if_eq_x_p:nn \{abc\} \{\tl_to_str:n \{abc\}\}
\end{verbatim}

is logically \textit{true}.

\str_case:nnTF \str_case:nnTF \str_case:nnTF \str_case:nnTF \str_case:nnTF \str_case:nnTF \str_case:nnTF

This function compares the \textit{test string} in turn with each of the \textit{string cases}. If the two are equal (as described for \texttt{\str_if_eq_x:nnTF} then the associated \textit{code} is left in the input stream. If any of the cases are matched, the \textit{true code} is also inserted into the input stream (after the code for the appropriate case), while if none match then the \textit{false code} is inserted. The function \texttt{\str_case:nn}, which does nothing if there is no match, is also available.

\str_case_x:nnF \str_case_x:nnF \str_case_x:nnF \str_case_x:nnF \str_case_x:nnF \str_case_x:nnF \str_case_x:nnF

This function compares the full expansion of the \textit{test string} in turn with the full expansion of the \textit{string cases}. If the two full expansions are equal (as described for \texttt{\str_if_eq_x:nnTF} then the associated \textit{code} is left in the input stream. If any of the cases are matched, the \textit{true code} is also inserted into the input stream (after the code for the appropriate case), while if none match then the \textit{false code} is inserted. The function \texttt{\str_case_x:nn}, which does nothing if there is no match, is also available. The \textit{test string} is expanded in each comparison, and must always yield the same result: for example, random numbers must not be used within this string.
2 String manipulation

\str_lower_case:n \str_upper_case:n \str_lower_case:f \str_upper_case:f
\str_lower_case:n \str_upper_case:n \str_lower_case:f \str_upper_case:f

Converts the input \langle tokens\rangle to their string representation, as described for \tl_to_str:n, and then to the lower or upper case representation using a one-to-one mapping as described by the Unicode Consortium file UnicodeData.txt.

These functions are intended for case changing programmatic data in places where upper/lower case distinctions are meaningful. One example would be automatically generating a function name from user input where some case changing is needed. In this situation the input is programmatic, not textual, case does have meaning and a language-independent one-to-one mapping is appropriate. For example

\cs_new_protected:Npn \myfunc:nn #1#2
{\cs_set_protected:cpn
 {user \str_upper_case:f {\tl_head:n {#1}} \str_lower_case:f {\tl_tail:n {#1}}}
 {#2}}

would be used to generate a function with an auto-generated name consisting of the upper case equivalent of the supplied name followed by the lower case equivalent of the rest of the input.

These functions should not be used for

- Caseless comparisons: use \str_fold_case:n for this situation (case folding is distinct from lower casing).
- Case changing text for typesetting: see the \tl_lower_case:n(n), \tl_upper_case:n(n) and \tl_mixed_case:n(n) functions which correctly deal with context-dependence and other factors appropriate to text case changing.

\TeXhackers note: As with all expl3 functions, the input supported by \str_fold_case:n is engine-native characters which are or interoperate with UTF-8. As such, when used with pdf\TeX only the Latin alphabet characters A–Z will be case-folded (i.e. the ASCII range which coincides with UTF-8). Full UTF-8 support is available with both Xe\TeX and Lua\TeX, subject only to the fact that Xe\TeX in particular has issues with characters of code above hexadecimal 0xFFF when interacting with \tl_to_str:n.
\str_fold_case:n \str_fold_case:n \{\{tokens\}\}

Converts the input \{tokens\} to their string representation, as described for \tl_to_str:n, and then folds the case of the resulting \{string\} to remove case information. The result of this process is left in the input stream.

String folding is a process used for material such as identifiers rather than for “text”. The folding provided by \str_fold_case:n follows the mappings provided by the Unicode Consortium, who state:

Case folding is primarily used for caseless comparison of text, such as identifiers in a computer program, rather than actual text transformation. Case folding in Unicode is based on the lowercase mapping, but includes additional changes to the source text to help make it language-insensitive and consistent. As a result, case-folded text should be used solely for internal processing and generally should not be stored or displayed to the end user.

The folding approach implemented by \str_fold_case:n follows the “full” scheme defined by the Unicode Consortium (e.g. SSfolds to SS). As case-folding is a language-insensitive process, there is no special treatment of Turkic input (i.e. I always folds to i and not to ķ).

\TeXhackers note: As with all expl3 functions, the input supported by \str_fold_case:n is engine-native characters which are or interoperate with UTF-8. As such, when used with pdf\TeX only the Latin alphabet characters A–Z will be case-folded (i.e. the ASCII range which coincides with UTF-8). Full UTF-8 support is available with both X\TeX and Lua\TeX, subject only to the fact that X\TeX in particular has issues with characters of code above hexadecimal 0xFFF when interacting with \tl_to_str:n.

2.1 Internal string functions

\__str_if_eq_x:nn \__str_if_eq_x:nn \{\{tl_1\}\} \{\{tl_2\}\}

Compares the full expansion of two \{token lists\} on a character by character basis, and is \texttt{true} if the two lists contain the same characters in the same order. Leaves 0 in the input stream if the condition is true, and +1 or -1 otherwise.

\__str_if_eq_x_return:nn \__str_if_eq_x_return:nn \{\{tl_1\}\} \{\{tl_2\}\}

Compares the full expansion of two \{token lists\} on a character by character basis, and is \texttt{true} if the two lists contain the same characters in the same order. Either \texttt{\prg_return_true:} or \texttt{\prg_return_false:} is then left in the input stream. This is a version of \str_if_eq_x:nn(TF) coded for speed.
Part XIII
The \l3seq package
Sequences and stacks

\LaTeXe{} implements a “sequence” data type, which contain an ordered list of entries which may contain any \emph{balanced text}. It is possible to map functions to sequences such that the function is applied to every item in the sequence.

Sequences are also used to implement stack functions in \LaTeXe{}. This is achieved using a number of dedicated stack functions.

1 Creating and initialising sequences

\begin{verbatim}
\seq_new:N \seq_new:c
\seq_clear:N \seq_clear:c \seq_gclear:N \seq_gclear:c
\seq_clear_new:N \seq_clear_new:c \seq_gclear_new:N \seq_gclear_new:c
\seq_set_eq:NN \seq_set_eq:cn \seq_gset_eq:NN \seq_gset_eq:cn
\seq_set_from_clist:NN \seq_set_from_clist:cn \seq_set_from_clist:Nn \seq_set_from_clist:cn
\seq_gset_from_clist:NN \seq_gset_from_clist:cn \seq_gset_from_clist:Nn \seq_gset_from_clist:cn
\end{verbatim}

\seq_new:N \seq_new:c

\verb|\seq_new:N \seq_new:c|

Creates a new \emph{sequence} or raises an error if the name is already taken. The declaration is global. The \emph{sequence} will initially contain no items.

\seq_clear:N \seq_clear:c \seq_gclear:N \seq_gclear:c

\verb|\seq_clear:N \seq_clear:c \seq_gclear:N \seq_gclear:c|

Clears all items from the \emph{sequence}.

\seq_clear_new:N \seq_clear_new:c \seq_gclear_new:N \seq_gclear_new:c

\verb|\seq_clear_new:N \seq_clear_new:c \seq_gclear_new:N \seq_gclear_new:c|

Ensures that the \emph{sequence} exists globally by applying \verb|\seq_new:N| if necessary, then applies \verb|\seq_(g)clear:N| to leave the \emph{sequence} empty.

\seq_set_eq:NN \seq_set_eq:cn \seq_gset_eq:NN \seq_gset_eq:cn

\verb|\seq_set_eq:NN \seq_set_eq:cn \seq_gset_eq:NN \seq_gset_eq:cn|

Sets the content of \emph{sequence\textsubscript{1}} equal to that of \emph{sequence\textsubscript{2}}.

\seq_set_from_clist:NN \seq_set_from_clist:cn \seq_set_from_clist:Nn \seq_set_from_clist:cn

\verb|\seq_set_from_clist:NN \seq_set_from_clist:cn \seq_set_from_clist:Nn \seq_set_from_clist:cn|

\verb|\seq_set_from_clist:NN \seq_set_from_clist:cn \seq_set_from_clist:Nn \seq_set_from_clist:cn|

\seq_gset_from_clist:NN \seq_gset_from_clist:cn \seq_gset_from_clist:Nn \seq_gset_from_clist:cn

\verb|\seq_gset_from_clist:NN \seq_gset_from_clist:cn \seq_gset_from_clist:Nn \seq_gset_from_clist:cn|

Converts the data in the \emph{comma list} into a \emph{sequence}: the original \emph{comma list} is unchanged.
Splits the \langle token list \rangle into \langle items \rangle separated by \langle delimiter \rangle, and assigns the result to the \langle sequence \rangle. Spaces on both sides of each \langle item \rangle are ignored, then one set of outer braces is removed (if any); this space trimming behaviour is identical to that of \texttt{l3clist} functions. Empty \langle items \rangle are preserved by \texttt{\seq_set_split:Nnn}, and can be removed afterwards using \texttt{\seq_remove_all:Nn \langle sequence \rangle \{ \}}. The \langle delimiter \rangle may not contain \{, \} or \# (assuming \TeX{}'s normal category code régime). If the \langle delimiter \rangle is empty, the \langle token list \rangle is split into \langle items \rangle as a \langle token list \rangle.

\begin{verbatim}
\seq_set_split:Nnn \seq_set_split:NnV \seq_gset_split:Nnn \seq_gset_split:NnV
\seq_concat:NNN \seq_concat:ccc \seq_gconcat:NNN \seq_gconcat:ccc
\seq_if_exist_p:N \seq_if_exist:NTF \seq_if_exist:N \seq_if_exist:cTF
\seq_put_left:Nn \seq_put_left:NV \seq_gput_left:Nn \seq_gput_left:NV
\seq_put_right:Nn \seq_put_right:NV \seq_gput_right:Nn \seq_gput_right:NV
\end{verbatim}

\begin{itemize}
  \item New: 2011-08-15
  \item Updated: 2012-07-02
\end{itemize}

2 Appending data to sequences

\texttt{\seq_put_left:Nn \langle sequence \rangle \{ \langle item \rangle \}}

Appends the \langle item \rangle to the left of the \langle sequence \rangle.

\texttt{\seq_put_right:Nn \langle sequence \rangle \{ \langle item \rangle \}}

Appends the \langle item \rangle to the right of the \langle sequence \rangle.

3 Recovering items from sequences

Items can be recovered from either the left or the right of sequences. For implementation reasons, the actions at the left of the sequence are faster than those acting on the right. These functions all assign the recovered material locally, \textit{i.e.} setting the \langle token list variable \rangle used with \texttt{\tl_set:Nn} and \texttt{\tl_gset:Nn}.
\seq_get_left:NN \seq_get_left:CN

Stores the left-most item from a \langle sequence \rangle in the \langle token list variable \rangle without removing it from the \langle sequence \rangle. The \langle token list variable \rangle is assigned locally. If \langle sequence \rangle is empty the \langle token list variable \rangle will contain the special marker \q_no_value.

\seq_get_right:NN \seq_get_right:CN

Stores the right-most item from a \langle sequence \rangle in the \langle token list variable \rangle without removing it from the \langle sequence \rangle. The \langle token list variable \rangle is assigned locally. If \langle sequence \rangle is empty the \langle token list variable \rangle will contain the special marker \q_no_value.

\seq_pop_left:NN \seq_pop_left:CN

Pops the left-most item from a \langle sequence \rangle into the \langle token list variable \rangle, i.e. removes the item from the sequence and stores it in the \langle token list variable \rangle. Both of the variables are assigned locally. If \langle sequence \rangle is empty the \langle token list variable \rangle will contain the special marker \q_no_value.

\seq_gpop_left:NN \seq_gpop_left:CN

Pops the left-most item from a \langle sequence \rangle into the \langle token list variable \rangle, i.e. removes the item from the sequence and stores it in the \langle token list variable \rangle. The \langle sequence \rangle is modified globally, while the assignment of the \langle token list variable \rangle is local. If \langle sequence \rangle is empty the \langle token list variable \rangle will contain the special marker \q_no_value.

\seq_pop_right:NN \seq_pop_right:CN

Pops the right-most item from a \langle sequence \rangle into the \langle token list variable \rangle, i.e. removes the item from the sequence and stores it in the \langle token list variable \rangle. Both of the variables are assigned locally. If \langle sequence \rangle is empty the \langle token list variable \rangle will contain the special marker \q_no_value.

\seq_gpop_right:NN \seq_gpop_right:CN

Pops the right-most item from a \langle sequence \rangle into the \langle token list variable \rangle, i.e. removes the item from the sequence and stores it in the \langle token list variable \rangle. The \langle sequence \rangle is modified globally, while the assignment of the \langle token list variable \rangle is local. If \langle sequence \rangle is empty the \langle token list variable \rangle will contain the special marker \q_no_value.
Indexing items in the \textit{sequence} from 1 at the top (left), this function will evaluate the \textit{integer expression} and leave the appropriate item from the sequence in the input stream. If the \textit{integer expression} is negative, indexing occurs from the bottom (right) of the sequence. When the \textit{integer expression} is larger than the number of items in the \textit{sequence} (as calculated by \texttt{\seq_count:N}) then the function will expand to nothing.

\textbf{\TeX}hackers note: The result is returned within the \texttt{\unexpanded} primitive (\texttt{\exp_not:n}), which means that the \textit{item} will not expand further when appearing in an \texttt{x}-type argument expansion.

4 Recovering values from sequences with branching

The functions in this section combine tests for non-empty sequences with recovery of an item from the sequence. They offer increased readability and performance over separate testing and recovery phases.

\begin{verbatim}
\seq_item:Nn \seq_item:cn \seq_item:cn
Indexing items in the \texttt{sequence} from 1 at the top (left), this function will evaluate the \texttt{integer expression} and leave the appropriate item from the sequence in the input stream. If the \texttt{integer expression} is negative, indexing occurs from the bottom (right) of the sequence. When the \texttt{integer expression} is larger than the number of items in the \texttt{sequence} (as calculated by \texttt{\seq_count:N}) then the function will expand to nothing.

\textbf{\TeX}hackers note: The result is returned within the \texttt{\unexpanded} primitive (\texttt{\exp_not:n}), which means that the \textit{item} will not expand further when appearing in an \texttt{x}-type argument expansion.

4 Recovering values from sequences with branching

The functions in this section combine tests for non-empty sequences with recovery of an item from the sequence. They offer increased readability and performance over separate testing and recovery phases.

\begin{verbatim}
\seq_item:Nn \seq_item:cn \seq_item:cn
\end{verbatim}
5 Modifying sequences

While sequences are normally used as ordered lists, it may be necessary to modify the content. The functions here may be used to update sequences, while retaining the order of the unaffected entries.

\seq_pop_right:NNTF \seq_pop_right:cNNTF
\seq_gpop_right:NNTF \seq_gpop_right:cNNTF
\seq_gpop_right:NN \seq_gpop_right:cN
New: 2012-05-19

If the \langle sequence \rangle is empty, leaves the \langle false code \rangle in the input stream. The value of the \langle token list variable \rangle is not defined in this case and should not be relied upon. If the \langle sequence \rangle is non-empty, pops the right-most item from a \langle sequence \rangle in the \langle token list variable \rangle, i.e. removes the item from a \langle sequence \rangle. Both the \langle sequence \rangle and the \langle token list variable \rangle are assigned locally.

\seq_gpop_right:NNTF \seq_gpop_right:cNNTF
\seq_gpop_right:NN \seq_gpop_right:cN
New: 2012-05-19

If the \langle sequence \rangle is empty, leaves the \langle false code \rangle in the input stream. The value of the \langle token list variable \rangle is not defined in this case and should not be relied upon. If the \langle sequence \rangle is non-empty, pops the right-most item from a \langle sequence \rangle in the \langle token list variable \rangle, i.e. removes the item from a \langle sequence \rangle. The \langle sequence \rangle is modified globally, while the \langle token list variable \rangle is assigned locally.

\seq_remove_duplicates:N \seq_remove_duplicates:c \seq_gremove_duplicates:N \seq_gremove_duplicates:c
\seq_remove_all:Nn \seq_remove_all:cn \seq_gremove_all:Nn \seq_gremove_all:cn
New: 2014-07-18

\seq_remove_duplicates:N \langle sequence \rangle
\seq_remove_duplicates:c \langle sequence \rangle
\seq_gremove_duplicates:N \langle sequence \rangle
\seq_gremove_duplicates:c \langle sequence \rangle

Removes duplicate items from the \langle sequence \rangle, leaving the left most copy of each item in the \langle sequence \rangle. The \langle item \rangle comparison takes place on a token basis, as for \tl_if_eq:nn(TF).

\textbf{TeXhackers note:} This function iterates through every item in the \langle sequence \rangle and does a comparison with the \langle items \rangle already checked. It is therefore relatively slow with large sequences.

\seq_remove_all:Nn \seq_remove_all:cn \seq_gremove_all:Nn \seq_gremove_all:cn
\seq_remove_all:Nn \langle sequence \rangle \{\langle item \rangle\}
\seq_remove_all:cn \langle sequence \rangle \{\langle item \rangle\}
\seq_gremove_all:Nn \langle sequence \rangle \{\langle item \rangle\}
\seq_gremove_all:cn \langle sequence \rangle \{\langle item \rangle\}

Removes every occurrence of \langle item \rangle from the \langle sequence \rangle. The \langle item \rangle comparison takes place on a token basis, as for \tl_if_eq:nn(TF).

\seq_reverse:N \seq_reverse:c \seq_greverse:N \seq_greverse:c
New: 2014-07-18

\seq_reverse:N \langle sequence \rangle
\seq_reverse:c \langle sequence \rangle
\seq_greverse:N \langle sequence \rangle
\seq_greverse:c \langle sequence \rangle

Reverses the order of the items stored in the \langle sequence \rangle.
6 Sequence conditionals

\seq_if_empty_p:N
\seq_if_empty:c
\seq_if_empty:N
\seq_if_empty:cTF

Tests if the \langle sequence \rangle is empty (containing no items).

\seq_if_empty_p:N \star
\seq_if_empty_p:c \star
\seq_if_empty:N \star
\seq_if_empty:cTF \star

\seq_if_in:NnTF
\seq_if_in:(NV|Nv|No|Nx|cn|cV|cv|co|cx)TF

Tests if the \langle item \rangle is present in the \langle sequence \rangle.

7 Mapping to sequences

\seq_map_function:NN
\seq_map_function:cN

Updated: 2012-06-29

Applies \langle function \rangle to every \langle item \rangle stored in the \langle sequence \rangle. The \langle function \rangle will receive one argument for each iteration. The \langle items \rangle are returned from left to right. The function \seq_map_inline:Nn is faster than \seq_map_function:NN for sequences with more than about 10 items. One mapping may be nested inside another.

\seq_map_function:NN \langle sequence \rangle \langle function \rangle
\seq_map_function:cN \langle sequence \rangle \langle function \rangle

Updated: 2012-06-29

\seq_map_inline:Nn
\seq_map_inline:cn

Updated: 2012-06-29

Applies \langle inline function \rangle to every \langle item \rangle stored within the \langle sequence \rangle. The \langle inline function \rangle should consist of code which will receive the \langle item \rangle as \#1. One in line mapping can be nested inside another. The \langle items \rangle are returned from left to right.

\seq_map_inline:Nn \langle sequence \rangle \langle inline function \rangle
\seq_map_inline:cn \langle sequence \rangle \langle inline function \rangle

Updated: 2012-06-29

\seq_map_variable:NNn
\seq_map_variable:(Ncn|cNn|ccn)

Updated: 2012-06-29

Stores each entry in the \langle sequence \rangle in turn in the \langle tl var. \rangle and applies the \langle function using tl var. \rangle. The \langle function \rangle will usually consist of code making use of the \langle tl var. \rangle, but this is not enforced. One variable mapping can be nested inside another. The \langle items \rangle are returned from left to right.
\seq_map_break: \seq_map_break:

Used to terminate a \seq_map_... function before all entries in the \langle sequence \rangle have been processed. This will normally take place within a conditional statement, for example

\seq_map_inline:Nn \l_my_seq
{\str_if_eq:nnTF { #1 } { bingo } \seq_map_break: }
{ \%
  Do something useful
}

Use outside of a \seq_map_... scenario will lead to low level \TeX{} errors.

\TeX{}hackers note: When the mapping is broken, additional tokens may be inserted by the internal macro \_\_prg_break_point:Nn before further items are taken from the input stream. This will depend on the design of the mapping function.

\seq_map_break:n
\seq_map_break:n \langle tokens \rangle

Updated: 2012-06-29

Used to terminate a \seq_map_... function before all entries in the \langle sequence \rangle have been processed, inserting the \langle tokens \rangle after the mapping has ended. This will normally take place within a conditional statement, for example

\seq_map_inline:Nn \l_my_seq
{\str_if_eq:nnTF { #1 } { bingo } \seq_map_break:n \langle tokens \rangle }
{ \%
  Do something useful
}

Use outside of a \seq_map_... scenario will lead to low level \TeX{} errors.

\TeX{}hackers note: When the mapping is broken, additional tokens may be inserted by the internal macro \_\_prg_break_point:Nn before the \langle tokens \rangle are inserted into the input stream. This will depend on the design of the mapping function.

\seq_count:N \seq_count:N \langle sequence \rangle

New: 2012-07-13

Leaves the number of items in the \langle sequence \rangle in the input stream as an \langle integer denotation \rangle. The total number of items in a \langle sequence \rangle will include those which are empty and duplicates, i.e. every item in a \langle sequence \rangle is unique.
8 Using the content of sequences directly

\seq_use:Nnnn \seq_use:cnnn

Places the contents of the \seqvar in the input stream, with the appropriate \seqsep between the items. Namely, if the sequence has more than two items, the \seqsep is placed between each pair of items except the last, for which the \seqfinal is used. If the sequence has exactly two items, then they are placed in the input stream separated by the \seqsep. If the sequence has a single item, it is placed in the input stream, and an empty sequence produces no output. An error will be raised if the variable does not exist or if it is invalid.

For example,

\seq_set_split:Nnn \l_tmpa_seq { | } { a | b | c | {de} | f }
\seq_use:Nnnn \l_tmpa_seq { ,and, }

will insert “a, b, c, de, and f” in the input stream. The first separator argument is not used in this case because the sequence has more than 2 items.

\TeXhackers note: The result is returned within the \unexpanded primitive (\exp_not:n), which means that the \items will not expand further when appearing in an x-type argument expansion.

9 Sequences as stacks

Sequences can be used as stacks, where data is pushed to and popped from the top of the sequence. (The left of a sequence is the top, for performance reasons.) The stack functions for sequences are not intended to be mixed with the general ordered data
functions detailed in the previous section: a sequence should either be used as an ordered data type or as a stack, but not in both ways.

\seq_get:NN \seq_get:cN
\seq_pop:NN \seq_pop:cN
\seq_gpop:NN \seq_gpop:cN

Updated: 2012-05-14
Updated: 2012-05-19
Updated: 2012-05-19

\seq_get:NNTF \seq_get:CN
\seq_pop:NNTF \seq_pop:CN
\seq_gpop:NNTF \seq_gpop:CN

New: 2012-05-14
Updated: 2012-05-19
Updated: 2012-05-19

\seq_push:Nn \seq_push:NV \seq_push:Nx \seq_push:V \seq_push:V \seq_push:cn \seq_push:co \seq_push:cx

Adds the \{item\} to the top of the \{sequence\}.  

118
10 Constant and scratch sequences

\c_empty_seq
New: 2012-07-02

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

\l_tmpa_seq
\l_tmpb_seq
New: 2012-04-26

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

11 Viewing sequences

\seq_show:N \seq_show:c
Updated: 2012-09-09

Displays the entries in the \texttt{\langle sequence\rangle} in the terminal.

12 Internal sequence functions

\s__seq

This scan mark (equal to \texttt{\scan_stop}) marks the beginning of a sequence variable.

\__seq_item:n
\__seq_item:n \texttt{\langle item\rangle}

The internal token used to begin each sequence entry. If expanded outside of a mapping or manipulation function, an error will be raised. The definition should always be set globally.

\__seq_push_item_def:n
\__seq_push_item_def:x
\__seq_push_item_def:n \texttt{\langle code\rangle}

Saves the definition of \texttt{\__seq_item:n} and redefines it to accept one parameter and expand to \texttt{\langle code\rangle}. This function should always be balanced by use of \texttt{\__seq_pop_item_def:n}.

\__seq_pop_item_def:
\__seq_pop_item_def:

Restores the definition of \texttt{\__seq_item:n} most recently saved by \texttt{\__seq_push_item_def:n}. This function should always be used in a balanced pair with \texttt{\__seq_push_item_def:n}.
Part XIV

The l3clist package
Comma separated lists

Comma lists contain ordered data where items can be added to the left or right end of the list. The resulting ordered list can then be mapped over using \clist_map_function:NN. Several items can be added at once, and spaces are removed from both sides of each item on input. Hence,

\clist_new:N \l_my_clist
\clist_put_left:Nn \l_my_clist { - a - , ~{b}~ } 
\clist_put_right:Nn \l_my_clist { ~ { c ~ } , d }

results in \l_my_clist containing a,{b},{c~},d. Comma lists cannot contain empty items, thus

\clist_clear_new:N \l_my_clist
\clist_put_right:Nn \l_my_clist { , ~ , , } 
\clist_if_empty:NTF \l_my_clist { true } { false }

will leave true in the input stream. To include an item which contains a comma, or starts or ends with a space, surround it with braces. The sequence data type should be preferred to comma lists if items are to contain {, }, or # (assuming the usual \TeX category codes apply).

1 Creating and initialising comma lists

\clist_new:N  \clist_new:c

\clist_new:N \clist_new:c (comma list)

Creates a new (comma list) or raises an error if the name is already taken. The declaration is global. The (comma list) will initially contain no items.

\clist_const:Nn \clist_const:cn \clist_const:cx

\clist_const:Nn \clist_const:cn \clist_const:cx (clist var) { (comma list) }

Creates a new constant (clist var) or raises an error if the name is already taken. The value of the (clist var) will be set globally to the (comma list).

\clist_clear:N \clist_clear:c \clist_gclear:N \clist_gclear:c

\clist_clear:N \clist_clear:c \clist_gclear:N \clist_gclear:c (comma list)

Clears all items from the (comma list).
\clist_clear_new:N
\clist_gclear_new:N
Ensures that the \textit{comma list} exists globally by applying \clist_new:N if necessary, then applies \clist_(g)clear:N to leave the list empty.

\clist_set_eq:NN
\clist_gset_eq:NN
Sets the content of \textit{comma list} equal to that of \textit{comma list}.

\clist_set_from_seq:NN
\clist_gset_from_seq:NN
New: 2011-09-06
Converts the data in the \textit{sequence} into a \textit{comma list}: the original \textit{sequence} is unchanged. Items which contain either spaces or commas are surrounded by braces.

\clist_concat:NNN
\clist_gconcat:NNN
Concatenates the content of \textit{comma list} and \textit{comma list} together and saves the result in \textit{comma list}. The items in \textit{comma list} will be placed at the left side of the new comma list.

\clist_if_exist_p:N
\clist_if_exist:NTF
\clist_if_exist:cTF
Tests whether the \textit{comma list} is currently defined. This does not check that the \textit{comma list} really is a comma list.

\clist_set:Nn
\clist_gset:Nn
New: 2011-09-06
Sets \textit{comma list} to contain the \textit{items}, removing any previous content from the variable. Spaces are removed from both sides of each item.

2 Adding data to comma lists
3 Modifying comma lists

While comma lists are normally used as ordered lists, it may be necessary to modify the content. The functions here may be used to update comma lists, while retaining the order of the unaffected entries.

\clist_remove_duplicates:N \clist_gremove_duplicates:N \clist_remove_duplicates:N (comma list)
\clist_remove_all:N \clist_gremove_all:N \clist_remove_all:N (comma list) {⟨item⟩,⟨item⟩,...,⟨item⟩}
\clist_remove_all:Nn \clist_gremove_all:Nn \clist_remove_all:cn \clist_gremove_all:cn
\clist_remove_duplicates:N \clist_gremove_duplicates:N \clist_remove_duplicates:c \clist_gremove_duplicates:c
\clist_remove_all:N \clist_gremove_all:N \clist_remove_all:cn \clist_gremove_all:cn

Removes duplicate items from the (comma list), leaving the left most copy of each item in the (comma list). The ⟨item⟩ comparison takes place on a token basis, as for \tl_if_eq:nn(TF).

\TeXhacks{This function iterates through every item in the (comma list) and does a comparison with the ⟨items⟩ already checked. It is therefore relatively slow with large comma lists. Furthermore, it will not work if any of the items in the (comma list) contains {, }, or # (assuming the usual \TeX category codes apply).}

\clist_remove_all:Nn \clist_gremove_all:Nn \clist_remove_all:cn \clist_gremove_all:cn

Removes every occurrence of ⟨item⟩ from the (comma list). The ⟨item⟩ comparison takes place on a token basis, as for \tl_if_eq:nn(TF).

\TeXhacks{The ⟨item⟩ may not contain {, }, or # (assuming the usual \TeX category codes apply).}
\clist_reverse:N \clist_reverse:c
\clist_greverse:N \clist_greverse:c

New: 2014-07-18

\clist_reverse:n \clist_reverse:n \langle comma list \rangle
Leaves the items in the \langle comma list \rangle in the input stream in reverse order. Braces and spaces are preserved by this process.

\TeXhackers note: The result is returned within \unexpanded, which means that the comma list will not expand further when appearing in an x-type argument expansion.

4 Comma list conditionals

\clist_if_empty_p:N \clist_if_empty:NTF
\clist_if_empty_p:c \clist_if_empty:cTF
\clist_if_empty:n \clist_if_empty:nTF

Tests if the \langle comma list \rangle is empty (containing no items).

\clist_if_empty:n \clist_if_empty:n \langle comma list \rangle
\clist_if_empty:n \langle comma list \rangle \{ \langle true code \rangle \} \{ \langle false code \rangle \}
Tests if the \langle comma list \rangle is empty (containing no items). The rules for space trimming are as for other n-type comma-list functions, hence the comma list \{ ~ , ~ , ~ \} (without outer braces) is empty, while \{ ~ , { } \} (without outer braces) contains one element, which happens to be empty: the comma-list is not empty.

\clist_if_in:NnTF \clist_if_in:nnTF
\clist_if_in:n \clist_if_in:n { \{ NV\No|cn|cV\co \} TF
\clist_if_in:n \clist_if_in:n \langle nV\no \rangle TF

Updated: 2011-09-06
Tests if the \langle item \rangle is present in the \langle comma list \rangle. In the case of an n-type \langle comma list \rangle, spaces are stripped from each item, but braces are not removed. Hence,

\clist_if_in:n \{ a , \{ b \}~ , \{ b \} , c \} \{ b \} \{ \text{true} \} \{ \text{false} \}
yields \text{false}.

\TeXhackers note: The \langle item \rangle may not contain { , }, or \# (assuming the usual \TeX category codes apply), and should not contain \, nor start or end with a space.
5 Mapping to comma lists

The functions described in this section apply a specified function to each item of a comma list.

When the comma list is given explicitly, as an n-type argument, spaces are trimmed around each item. If the result of trimming spaces is empty, the item is ignored. Otherwise, if the item is surrounded by braces, one set is removed, and the result is passed to the mapped function. Thus, if your comma list that is being mapped is \{a/uni2423,/uni2423{{b}/uni2423},/uni2423,{},/uni2423{c},\} then the arguments passed to the mapped function are ‘a’, ‘{b}/uni2423’, an empty argument, and ‘c’.

When the comma list is given as an N-type argument, spaces have already been trimmed on input, and items are simply stripped of one set of braces if any. This case is more efficient than using n-type comma lists.

\clist_map_function:NN (comma list) (function)
Applies \langle function \rangle to every \langle item \rangle stored in the \langle comma list \rangle. The \langle function \rangle will receive one argument for each iteration. The \langle items \rangle are returned from left to right. The function \clist_map_inline:Nn is in general more efficient than \clist_map_function:NN. One mapping may be nested inside another.

\clist_map_inline:Nn (comma list) {\langle inline function \rangle}
Applies \langle inline function \rangle to every \langle item \rangle stored within the \langle comma list \rangle. The \langle inline function \rangle should consist of code which will receive the \langle item \rangle as #1. One in line mapping can be nested inside another. The \langle items \rangle are returned from left to right.

\clist_map_variable:NNn (comma list) {\langle tl var. \rangle} {\langle function using tl var. \rangle}
Stores each entry in the \langle comma list \rangle in turn in the \langle tl var. \rangle and applies the \langle function using tl var. \rangle. The \langle function \rangle will usually consist of code making use of the \langle tl var. \rangle, but this is not enforced. One variable mapping can be nested inside another. The \langle items \rangle are returned from left to right.
\clist_map_break: \star \clist_map_break:  

Updated: 2012-06-29  

Used to terminate a \clist_map... function before all entries in the \langle comma list \rangle have been processed. This will normally take place within a conditional statement, for example

\clist_map_inline:Nn \l_my_clist  
{  
  \str_if_eq:nnTF { #1 } { bingo }  
  { \clist_map_break: }  
  {  
    \% Do something useful  
  }  
}

Use outside of a \clist_map... scenario will lead to low level \TeX errors.

\TeXhacksnote: When the mapping is broken, additional tokens may be inserted by the internal macro \_\_prg_break_point:Nn before further items are taken from the input stream. This will depend on the design of the mapping function.

\clist_map_break:n \star \clist_map_break:n \{\langle tokens \rangle\}  

Updated: 2012-06-29  

Used to terminate a \clist_map... function before all entries in the \langle comma list \rangle have been processed, inserting the \langle tokens \rangle after the mapping has ended. This will normally take place within a conditional statement, for example

\clist_map_inline:Nn \l_my_clist  
{  
  \str_if_eq:nnTF { #1 } { bingo }  
  { \clist_map_break:n \{\langle tokens \rangle\} }  
  {  
    \% Do something useful  
  }  
}

Use outside of a \clist_map... scenario will lead to low level \TeX errors.

\TeXhacksnote: When the mapping is broken, additional tokens may be inserted by the internal macro \_\_prg_break_point:Nn before the \langle tokens \rangle are inserted into the input stream. This will depend on the design of the mapping function.

\clist_count:N \star \clist_count:c \star \clist_count:n \star  

Updated: 2012-07-13  

\clist_count:N \langle comma list \rangle  

Leaves the number of items in the \langle comma list \rangle in the input stream as an \langle integer denotation \rangle. The total number of items in a \langle comma list \rangle will include those which are duplicates, \textit{i.e.} every item in a \langle comma list \rangle is unique.
6 Using the content of comma lists directly

\clist_use:Nnnn \clist_use:cn

Places the contents of the \clist var in the input stream, with the appropriate \langle separator \rangle between the items. Namely, if the comma list has more than two items, the \langle separator between more than two \rangle is placed between each pair of items except the last, for which the \langle separator between final two \rangle is used. If the comma list has exactly two items, then they are placed in the input stream separated by the \langle separator between two \rangle. If the comma list has a single item, it is placed in the input stream, and a comma list with no items produces no output. An error will be raised if the variable does not exist or if it is invalid.

For example,

\clist_set:Nn \l_tmpa_clist { a, b, , c, {de}, f }
\clist_use:Nnnn \l_tmpa_clist { ~and~ } { ,~ } { ,~and~ }

will insert “a, b, c, de, and f” in the input stream. The first separator argument is not used in this case because the comma list has more than 2 items.

\TeXhackers note: The result is returned within the \unexpanded primitive (\exp_not:n), which means that the \langle items \rangle will not expand further when appearing in an x-type argument expansion.

7 Comma lists as stacks

Comma lists can be used as stacks, where data is pushed to and popped from the top of the comma list. (The left of a comma list is the top, for performance reasons.) The
stack functions for comma lists are not intended to be mixed with the general ordered data functions detailed in the previous section: a comma list should either be used as an ordered data type or as a stack, but not in both ways.

\clist_get:NN \clist_get:cN
Updated: 2012-05-14

Stores the left-most item from a (comma list) in the (token list variable) without removing it from the (comma list). The (token list variable) is assigned locally. If the (comma list) is empty the (token list variable) will contain the marker value \q_no_value.

\clist_get:NNTF \clist_get:cNTF
New: 2012-05-14

If the (comma list) is empty, leaves the (false code) in the input stream. The value of the (token list variable) is not defined in this case and should not be relied upon. If the (comma list) is non-empty, stores the top item from the (comma list) in the (token list variable) without removing it from the (comma list). The (token list variable) is assigned locally.

\clist_pop:NN \clist_pop:cN
Updated: 2011-09-06

Pops the left-most item from a (comma list) into the (token list variable), i.e. removes the item from the comma list and stores it in the (token list variable). Both of the variables are assigned locally.

\clist_gpop:NN \clist_gpop:cN

Pops the left-most item from a (comma list) into the (token list variable), i.e. removes the item from the comma list and stores it in the (token list variable). The (comma list) is modified globally, while the assignment of the (token list variable) is local.

\clist_pop:NNTF \clist_pop:cNTF
New: 2012-05-14

If the (comma list) is empty, leaves the (false code) in the input stream. The value of the (token list variable) is not defined in this case and should not be relied upon. If the (comma list) is non-empty, pops the top item from the (comma list) in the (token list variable), i.e. removes the item from the (comma list). Both the (comma list) and the (token list variable) are assigned locally.

\clist_gpop:NNTF \clist_gpop:cNTF
New: 2012-05-14

If the (comma list) is empty, leaves the (false code) in the input stream. The value of the (token list variable) is not defined in this case and should not be relied upon. If the (comma list) is non-empty, pops the top item from the (comma list) in the (token list variable), i.e. removes the item from the (comma list). The (comma list) is modified globally, while the (token list variable) is assigned locally.
\clist_push:Nn \clist_push:NVNoNxcnVcVco\clist_gpush:Nn

Adds the \{\texttt{\items}\} to the top of the \texttt{\comma list}. Spaces are removed from both sides of each item.

8 Using a single item

\clist_item:Nn \clist_item:cn \clist_item:nn

Indexing items in the \texttt{\comma list} from 1 at the top (left), this function will evaluate the \texttt{\integer expression} and leave the appropriate item from the comma list in the input stream. If the \texttt{\integer expression} is negative, indexing occurs from the bottom (right) of the comma list. When the \texttt{\integer expression} is larger than the number of items in the \texttt{\comma list} (as calculated by \clist_count:N) then the function will expand to nothing.

\textit{\LaTeX}hackers note: The result is returned within the \texttt{\unexpanded} primitive (\exp_not:n), which means that the \texttt{\item} will not expand further when appearing in an \texttt{x}-type argument expansion.

9 Viewing comma lists

\clist_show:N \clist_show:c

Displays the entries in the \texttt{\comma list} in the terminal.

\clist_show:n

Displays the entries in the comma list in the terminal.

10 Constant and scratch comma lists

\c_empty_clist

Constant that is always empty.

\l_tmpa_clist \l_tmpb_clist

Scratch comma lists for local assignment. These are never used by the kernel code, and so are safe for use with any \LaTeX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.
\g_tmpa_clist \g_tmpb_clist

Scratch comma lists for global assignment. These are never used by the kernel code, and so are safe for use with any \TEX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.
Part XV

The \texttt{l3prop} package

Property lists

\LaTeX{}3 implements a “property list” data type, which contain an unordered list of entries each of which consists of a \langle key Deb\rangle and an associated \langle value Deb\rangle. The \langle key Deb\rangle and \langle value Deb\rangle may both be any \langle balanced text Deb\rangle. It is possible to map functions to property lists such that the function is applied to every key–value pair within the list.

Each entry in a property list must have a unique \langle key Deb\rangle: if an entry is added to a property list which already contains the \langle key Deb\rangle then the new entry will overwrite the existing one. The \langle keys Deb\rangle are compared on a string basis, using the same method as \texttt{\string_if_eq:nn}.\texttt{\string_if_eq:nn}.

Property lists are intended for storing key-based information for use within code. This is in contrast to key–value lists, which are a form of \textit{input} parsed by the \texttt{keys} module.

1 Creating and initialising property lists

\begin{verbatim}
\prop_new:N \prop_new:N \prop_new:c
\end{verbatim}

Creates a new \langle property list Deb\rangle or raises an error if the name is already taken. The declaration is global. The \langle property list Deb\rangle will initially contain no entries.

\begin{verbatim}
\prop_clear:N \prop_clear:N \prop_clear:c \prop_clear:c
\end{verbatim}

Clears all entries from the \langle property list Deb\rangle.

\begin{verbatim}
\prop_clear_new:N \prop_clear_new:N \prop_clear_new:c \prop_clear_new:c
\end{verbatim}

Ensures that the \langle property list Deb\rangle exists globally by applying \texttt{\prop_new:N} if necessary, then applies \texttt{\prop_\texttt{(g)}clear:N} to leave the list empty.

\begin{verbatim}
\prop_set_eq:NN \prop_set_eq:NN \prop_set_eq:NN \prop_set_eq:NN \prop_set_eq:NN \prop_set_eq:NN \prop_set_eq:NN \prop_set_eq:NN \prop_set_eq:NN \prop_set_eq:NN \prop_set_eq:NN
\end{verbatim}

Sets the content of \langle property list Deb\rangle equal to that of \langle property list Deb\rangle.
2 Adding entries to property lists

\prop_put:NnN
\prop_put:(NnV|Nno|Nnx|NVn|NVV|Non|Noo|cnN|cnV|cno|cVn|cV|cn|coo)
\prop_put_if_new:cnn
\prop_put_if_new:Nnn
\prop_put_if_new:cnn
\prop_gput_if_new:cnn
\prop_gput_if_new:Nnn
\prop_gput_if_new:cnn
\prop_gput:Nnn
\prop_gput:Nnn
\prop_gpop:NnN
\prop_gpop:(NoN|cnN|cV|coo)

Updated: 2012-07-09

Adds an entry to the \langle property list \rangle which may be accessed using the \langle key \rangle and which has \langle value \rangle. Both the \langle key \rangle and \langle value \rangle may contain any \langle balanced text \rangle. The \langle key \rangle is stored after processing with \texttt{\tl_to_str:n}, meaning that category codes are ignored. If the \langle key \rangle is already present in the \langle property list \rangle, the existing entry is overwritten by the new \langle value \rangle.

\prop_put_if_new:NnN
\prop_put_if_new:cnn
\prop_put_if_new:NnN
\prop_put_if_new:cnn
\prop_pop:NnN
\prop_pop:(NoN|cnN|cV|coo)

Updated: 2011-08-28

If the \langle key \rangle is present in the \langle property list \rangle then no action is taken. If the \langle key \rangle is not present in the \langle property list \rangle then a new entry is added. Both the \langle key \rangle and \langle value \rangle may contain any \langle balanced text \rangle. The \langle key \rangle is stored after processing with \texttt{\tl_to_str:n}, meaning that category codes are ignored.

3 Recovering values from property lists

\prop_get:NnN
\prop_get:(NVn|Nno|cnN|cVn|cn|coo)

Updated: 2011-08-28

Recovers the \langle value \rangle stored with \langle key \rangle from the \langle property list \rangle, and places this in the \langle token list variable \rangle. If the \langle key \rangle is not found in the \langle property list \rangle then the \langle token list variable \rangle will contain the special marker \texttt{\q_no_value}. The \langle token list variable \rangle is set within the current \texttt{\TeX} group. See also \prop_get:NnNTF.

\prop_pop:NnN
\prop_pop:(NoN|cnN|cV|coo)

Updated: 2011-08-18

Recovers the \langle value \rangle stored with \langle key \rangle from the \langle property list \rangle, and places this in the \langle token list variable \rangle. If the \langle key \rangle is not found in the \langle property list \rangle then the \langle token list variable \rangle will contain the special marker \texttt{\q_no_value}. The \langle key \rangle and \langle value \rangle are then deleted from the property list. Both assignments are local. See also \prop_pop:NnNTF.

\prop_gpop:NnN
\prop_gpop:(NoN|cnN|cV|coo)

Updated: 2011-08-18

Recovers the \langle value \rangle stored with \langle key \rangle from the \langle property list \rangle, and places this in the \langle token list variable \rangle. If the \langle key \rangle is not found in the \langle property list \rangle then the \langle token list variable \rangle will contain the special marker \texttt{\q_no_value}. The \langle key \rangle and \langle value \rangle are then deleted from the property list. The \langle property list \rangle is modified globally, while the assignment of the \langle token list variable \rangle is local. See also \prop_gpop:NnNTF.
\prop_item:Nn \prop_item:cn

Expands to the \langle value \rangle corresponding to the \langle key \rangle in the \langle property list \rangle. If the \langle key \rangle is missing, this has an empty expansion.

\TeX\hackers\ note: This function is slower than the non-expandable analogue \prop_get:NnN. The result is returned within the \unexpanded primitive (\exp_not:n), which means that the \langle value \rangle will not expand further when appearing in an x-type argument expansion.

4 Modifying property lists

\prop_remove:Nn \prop_remove:(NV|cn|cV) \prop_gremove:Nn \prop_gremove:(NV|cn|cV)

Removes the entry listed under \langle key \rangle from the \langle property list \rangle. If the \langle key \rangle is not found in the \langle property list \rangle no change occurs, i.e there is no need to test for the existence of a key before deleting it.

5 Property list conditionals

\prop_if_exist_p:N \prop_if_exist_p:c \prop_if_exist:NTF \prop_if_exist:cTF

Tests whether the \langle property list \rangle is currently defined. This does not check that the \langle property list \rangle really is a property list variable.

\prop_if_empty_p:N \prop_if_empty_p:c \prop_if_empty:NTF \prop_if_empty:cTF

Tests if the \langle property list \rangle is empty (containing no entries).

\prop_if_in_p:N\prop_if_in:(NV|No|cn|cV|co) \prop_if_in_p:NTF \prop_if_in:cTF

Tests if the \langle key \rangle is present in the \langle property list \rangle, making the comparison using the method described by \str_if_eq:nnTF.

\TeX\hackers\ note: This function iterates through every key–value pair in the \langle property list \rangle and is therefore slower than using the non-expandable \prop_get:NnNTF.
6 Recovering values from property lists with branching

The functions in this section combine tests for the presence of a key in a property list with recovery of the associated valued. This makes them useful for cases where different cases follow dependent on the presence or absence of a key in a property list. They offer increased readability and performance over separate testing and recovery phases.

\prop_get:NnNTF \prop_get:(NVN)NnN[cVN]cN\prop_get:NnN \prop_get:NnN \prop_get:NnN \prop_get:NnN \prop_get:NnN \prop_get:NnN \prop_get:NnN \prop_get:NnN

If the \textit{key} is not present in the \textit{property list}, leaves the \textit{false code} in the input stream. The value of the \textit{token list variable} is not defined in this case and should not be relied upon. If the \textit{key} is present in the \textit{property list}, stores the corresponding \textit{value} in the \textit{token list variable} without removing it from the \textit{property list}, then leaves the \textit{true code} in the input stream. The \textit{token list variable} is assigned locally.

\prop_pop:NnNTF \prop_pop:NnN \prop_pop:NnN \prop_pop:NnN \prop_pop:NnN \prop_pop:NnN \prop_pop:NnN \prop_pop:NnN

If the \textit{key} is not present in the \textit{property list}, leaves the \textit{false code} in the input stream. The value of the \textit{token list variable} is not defined in this case and should not be relied upon. If the \textit{key} is present in the \textit{property list}, pops the corresponding \textit{value} in the \textit{token list variable}, \textit{i.e.} removes the item from the \textit{property list}. Both the \textit{property list} and the \textit{token list variable} are assigned locally.

\prop_gpop:NnNTF \prop_gpop:NnN \prop_gpop:NnN \prop_gpop:NnN \prop_gpop:NnN \prop_gpop:NnN \prop_gpop:NnN \prop_gpop:NnN

If the \textit{key} is not present in the \textit{property list}, leaves the \textit{false code} in the input stream. The value of the \textit{token list variable} is not defined in this case and should not be relied upon. If the \textit{key} is present in the \textit{property list}, pops the corresponding \textit{value} in the \textit{token list variable}, \textit{i.e.} removes the item from the \textit{property list}. The \textit{property list} is modified globally, while the \textit{token list variable} is assigned locally.

7 Mapping to property lists

\prop_map_function:NN \prop_map_function:cN \prop_map_function:NN \prop_map_function:NN \prop_map_function:NN

Applies \textit{function} to every \textit{entry} stored in the \textit{property list}. The \textit{function} will receive two argument for each iteration: the \textit{key} and associated \textit{value}. The order in which \textit{entries} are returned is not defined and should not be relied upon.
\prop_map_inline:Nn \prop_map_inline:Nn \prop_map_inline:cn
Applies \textit{inline function} to every \textit{entry} stored within the \textit{property list}. The \textit{inline function} should consist of code which will receive the \textit{key} as \#1 and the \textit{value} as \#2. The order in which \textit{entries} are returned is not defined and should not be relied upon.

\prop_map_break: \prop_map_break:
Used to terminate a \texttt{\prop_map...} function before all entries in the \textit{property list} have been processed. This will normally take place within a conditional statement, for example

\begin{verbatim}
\prop_map_inline:Nn \l_my_prop
{  \str_if_eq:nTF { #1 } { bingo } { \prop_map_break: } {    \% Do something useful}
}
\end{verbatim}

Use outside of a \texttt{\prop_map...} scenario will lead to low level \TeX{} errors.

\prop_map_break:n \prop_map_break:n
Used to terminate a \texttt{\prop_map...} function before all entries in the \textit{property list} have been processed, inserting the \textit{tokens} after the mapping has ended. This will normally take place within a conditional statement, for example

\begin{verbatim}
\prop_map_inline:Nn \l_my_prop
{  \str_if_eq:nTF { #1 } { bingo } { \prop_map_break:n { <tokens> } } {    \% Do something useful}
}
\end{verbatim}

Use outside of a \texttt{\prop_map...} scenario will lead to low level \TeX{} errors.

8 Viewing property lists

\prop_show:N \prop_show:N \prop_show:c
Displays the entries in the \textit{property list} in the terminal.
9 Scratch property lists

Scratch property lists for local assignment. These are never used by the kernel code, and so are safe for use with any \LaTeX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

\l_tmpa_prop
\l_tmpb_prop
New: 2012-06-23

Scratch property lists for global assignment. These are never used by the kernel code, and so are safe for use with any \LaTeX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

\g_tmpa_prop
\g_tmpb_prop
New: 2012-06-23

10 Constants

\c_empty_prop
A permanently-empty property list used for internal comparisons.

11 Internal property list functions

\s__prop
The internal token used at the beginning of property lists. This is also used after each \texttt{(key)} (see \texttt{\_prop_pair:wn}).

\__prop_pair:wn \texttt{(key)} \s__prop \{\texttt{(item)}\}

The internal token used to begin each key–value pair in the property list. If expanded outside of a mapping or manipulation function, an error will be raised. The definition should always be set globally.

\l__prop_internal_tl
Token list used to store new key–value pairs to be inserted by functions of the \texttt{\prop_-put:Nnn} family.

\__prop_split:NnTF
\texttt{\_prop_split:NnTF \{property list\} \{\texttt{(key)}\} \{\texttt{(true code)}\} \{\texttt{(false code)}\}}

Splits the \texttt{(property list)} at the \texttt{(key)}, giving three token lists: the \texttt{(extract)} of \texttt{(property list)} before the \texttt{(key)}, the \texttt{(value)} associated with the \texttt{(key)} and the \texttt{(extract)} of the \texttt{(property list)} after the \texttt{(value)}. Both \texttt{(extracts)} retain the internal structure of a property list, and the concatenation of the two \texttt{(extracts)} is a property list. If the \texttt{(key)} is present in the \texttt{(property list)} then the \texttt{(true code)} is left in the input stream, with \#1, \#2, and \#3 replaced by the first \texttt{(extract)}, the \texttt{(value)}, and the second extract. If the \texttt{(key)} is not present in the \texttt{(property list)} then the \texttt{(false code)} is left in the input stream, with no trailing material. Both \texttt{(true code)} and \texttt{(false code)} are used in the replacement text of a macro defined internally, hence macro parameter characters should be doubled, except \#1, \#2, and \#3 which stand in the \texttt{(true code)} for the three extracts from the property list. The \texttt{(key)} comparison takes place as described for \texttt{\str_if_eq:nn}.
The l3box package

Boxes

There are three kinds of box operations: horizontal mode denoted with prefix \hbox_, vertical mode with prefix \vbox_, and the generic operations working in both modes with prefix \box_.

1 Creating and initialising boxes

\\box_new:N \box_new:N \langle box \rangle \box_new:N \box_new:c

Creates a new \langle box \rangle or raises an error if the name is already taken. The declaration is global. The \langle box \rangle will initially be void.

\box_clear:N \box_clear:N \langle box \rangle \box_clear:N \box_clear:c

Clears the content of the \langle box \rangle by setting the box equal to \c_void_box.

\box_clear_new:N \box_clear_new:N \langle box \rangle \box_clear_new:N \box_clear_new:c

Ensures that the \langle box \rangle exists globally by applying \box_new:N if necessary, then applies \box_(g)clear:N to leave the \langle box \rangle empty.

\box_set_eq:NN \box_set_eq:NN \langle box_1 \rangle \langle box_2 \rangle \box_set_eq:NN \box_set_eq:(cN|Nc|cc)

Sets the content of \langle box_1 \rangle equal to that of \langle box_2 \rangle.

\box_set_eq_clear:NN \box_set_eq_clear:NN \langle box_1 \rangle \langle box_2 \rangle \box_set_eq_clear:NN \box_set_eq_clear:(cN|Nc|cc)

Sets the content of \langle box_1 \rangle within the current \TeX group equal to that of \langle box_2 \rangle, then clears \langle box_2 \rangle globally.

\box_gset_eq:NN \box_gset_eq:NN \langle box_1 \rangle \langle box_2 \rangle \box_gset_eq:NN \box_gset_eq:(cN|Nc|cc)

Sets the content of \langle box_1 \rangle equal to that of \langle box_2 \rangle, then clears \langle box_2 \rangle. These assignments are global.
2 Using boxes

\box_use:N \box_use:c
Inserts the current content of the \textit{box} onto the current list for typesetting.

\textbf{\LaTeX} hackers note: This is the \LaTeX\ primitive \texttt{\copy}.

\box_use_clear:N \box_use_clear:c
Inserts the current content of the \textit{box} onto the current list for typesetting, then globally clears the content of the \textit{box}.

\textbf{\LaTeX} hackers note: This is the \LaTeX\ primitive \texttt{\box}.

\box_move_right:nn \box_move_left:nn
This function operates in vertical mode, and inserts the material specified by the \textit{box function} such that its reference point is displaced horizontally by the given \texttt{dimexpr} from the reference point for typesetting, to the right or left as appropriate. The \textit{box function} should be a box operation such as \texttt{\box_use:N \textbackslash box} or a “raw” box specification such as \texttt{\vbox:n \{ xyz \}}.

\box_move_up:nn \box_move_down:nn
This function operates in horizontal mode, and inserts the material specified by the \textit{box function} such that its reference point is displaced vertically by the given \texttt{dimexpr} from the reference point for typesetting, up or down as appropriate. The \textit{box function} should be a box operation such as \texttt{\box_use:N \textbackslash box} or a “raw” box specification such as \texttt{\vbox:n \{ xyz \}}.

3 Measuring and setting box dimensions

\box_dp:N \box_dp:c
Calculates the depth (below the baseline) of the \textit{box} in a form suitable for use in a \textit{dimension expression}.

\textbf{\LaTeX} hackers note: This is the \LaTeX\ primitive \texttt{\dp}.
Calculates the height (above the baseline) of the \langle box \rangle in a form suitable for use in a \langle dimension expression \rangle.

**TeXhackers note:** This is the \TeX primitive \texttt{ht}.

Calculates the width of the \langle box \rangle in a form suitable for use in a \langle dimension expression \rangle.

**TeXhackers note:** This is the \TeX primitive \texttt{wd}.

\texttt{\box_set_dp:Nn \langle box \rangle \{\langle dimension expression \rangle\}}

Set the depth (below the baseline) of the \langle box \rangle to the value of the \{\langle dimension expression \rangle\}. This is a global assignment.

\texttt{\box_set_ht:Nn \langle box \rangle \{\langle dimension expression \rangle\}}

Set the height (above the baseline) of the \langle box \rangle to the value of the \{\langle dimension expression \rangle\}. This is a global assignment.

\texttt{\box_set_wd:Nn \langle box \rangle \{\langle dimension expression \rangle\}}

Set the width of the \langle box \rangle to the value of the \{\langle dimension expression \rangle\}. This is a global assignment.

\section{Box conditionals}

\texttt{\box_if_empty_p:N \langle box \rangle \star \box_if_empty_p:c \star \box_if_empty:NTF \langle box \rangle \{\langle true code \rangle\} \{\langle false code \rangle\}}

Tests if \langle box \rangle is a empty (equal to \texttt{c_empty_box}).

\texttt{\box_if_horizontal_p:N \langle box \rangle \star \box_if_horizontal:p \star \box_if_horizontal:NTF \langle box \rangle \{\langle true code \rangle\} \{\langle false code \rangle\}}

Tests if \langle box \rangle is a horizontal box.

\texttt{\box_if_vertical_p:N \langle box \rangle \star \box_if_vertical:p \star \box_if_vertical:NTF \langle box \rangle \{\langle true code \rangle\} \{\langle false code \rangle\}}

Tests if \langle box \rangle is a vertical box.
5 The last box inserted

\texttt{\textbackslash box\_set\_to\_last:N} (box)

Sets the \texttt{(box)} equal to the last item (box) added to the current partial list, removing the item from the list at the same time. When applied to the main vertical list, the \texttt{(box)} will always be void as it is not possible to recover the last added item.

6 Constant boxes

\texttt{\textbackslash c\_empty\_box}

This is a permanently empty box, which is neither set as horizontal nor vertical.

7 Scratch boxes

\texttt{\textbackslash l\_tmpa\_box} \texttt{\textbackslash l\_tmpb\_box}

Scratch boxes for local assignment. These are never used by the kernel code, and so are safe for use with any \LaTeX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

\texttt{\textbackslash g\_tmpa\_box} \texttt{\textbackslash g\_tmpb\_box}

Scratch boxes for global assignment. These are never used by the kernel code, and so are safe for use with any \LaTeX3-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

8 Viewing box contents

\texttt{\textbackslash box\_show:N} (box)

Shows full details of the content of the \texttt{(box)} in the terminal.

\texttt{\textbackslash box\_show:Nn} (box) \texttt{\textbackslash intexpr\textsubscript{1}} \texttt{\textbackslash intexpr\textsubscript{2}}

Display the contents of \texttt{(box)} in the terminal, showing the first \texttt{(intexpr\textsubscript{1})} items of the box, and descending into \texttt{(intexpr\textsubscript{2})} group levels.

\texttt{\textbackslash box\_log:N} (box)

Writes full details of the content of the \texttt{(box)} to the log.
\box_log:Nnn \box_log:Nnn \text{box} \langle \text{intexpr}_1 \rangle \langle \text{intexpr}_2 \rangle

Writes the contents of \text{box} to the log, showing the first \text{intexpr}_1 items of the box, and descending into \text{intexpr}_2 group levels.

## 9 Horizontal mode boxes

\hbox:n \{\text{contents}\}

Typesets the \text{contents} into a horizontal box of natural width and then includes this box in the current list for typesetting.

\textbf{\TeX} hackers note: This is the \TeX primitive \verb\hbox.  

\hbox_to_wd:nn \{\text{dimexpr}\} \{\text{contents}\}

Typesets the \text{contents} into a horizontal box of width \text{dimexpr} and then includes this box in the current list for typesetting.

\hbox_to_zero:n \{\text{contents}\}

Typesets the \text{contents} into a horizontal box of zero width and then includes this box in the current list for typesetting.

\hbox_set:Nn \hbox_gset:Nn
\hbox_set:cn \hbox_gset:cn

Typesets the \text{contents} at natural width and then stores the result inside the \text{box}.

\hbox_set_to_wd:Nnn \hbox_gset_to_wd:Nnn \{\text{dimexpr}\} \{\text{contents}\}

Typesets the \text{contents} to the width given by the \text{dimexpr} and then stores the result inside the \text{box}.

\hbox_overlap_right:n \hbox_overlap_left:n

Typesets the \text{contents} into a horizontal box of zero width such that material will protrude to the right of the insertion point.

Typesets the \text{contents} into a horizontal box of zero width such that material will protrude to the left of the insertion point.
\hbox_set:Nw \hbox_set:cw \hbox_gset:Nw \hbox_gset:cw
\hbox_unpack:N \hbox_unpack:c
\hbox_unpack_clear:N \hbox_unpack_clear:c
\hbox_set:Nw \hbox_set:cw \hbox_gset:Nw \hbox_gset:cw
\hbox_set:Nw \langle \text{contents} \rangle \hbox_set_end:

Typesets the $\langle \text{contents} \rangle$ at natural width and then stores the result inside the $\langle \text{box} \rangle$. In contrast to $\hbox_set:Nn$ this function does not absorb the argument when finding the $\langle \text{content} \rangle$, and so can be used in circumstances where the $\langle \text{content} \rangle$ may not be a simple argument.

\hbox_set:Nw \hbox_set:cw \hbox_gset:Nw \hbox_gset:cw
\hbox_unpack:N \langle \text{box} \rangle
\hbox_unpack:N \langle \text{box} \rangle
\hbox_unpack_clear:N \langle \text{box} \rangle
\hbox_unpack_clear:c

Unpacks the content of the horizontal $\langle \text{box} \rangle$, retaining any stretching or shrinking applied when the $\langle \text{box} \rangle$ was set.

**\TeXhacker** note: This is the \TeX primitive \unhcopy.

\vbox:n \{\langle \text{contents} \rangle\}
\vbox:n \{\langle \text{contents} \rangle\}

Typesets the $\langle \text{contents} \rangle$ into a vertical box of natural height and includes this box in the current list for typesetting.

**\TeXhacker** note: This is the \TeX primitive \vbox.

\vbox_top:n \{\langle \text{contents} \rangle\}
\vbox_top:n \{\langle \text{contents} \rangle\}

Typesets the $\langle \text{contents} \rangle$ into a vertical box of natural height and includes this box in the current list for typesetting. The baseline of the box will be equal to that of the first item added to the box.

**\TeXhacker** note: This is the \TeX primitive \vtop.

10 Vertical mode boxes

Vertical boxes inherit their baseline from their contents. The standard case is that the baseline of the box is at the same position as that of the last item added to the box. This means that the box will have no depth unless the last item added to it had depth. As a result most vertical boxes have a large height value and small or zero depth. The exception are $\_\text{top}$ boxes, where the reference point is that of the first item added. These tend to have a large depth and small height, although the latter will typically be non-zero.
\vbox_to_ht:nn \vbox_to_ht:nn ⟨dimexpr⟩ ⟨contents⟩
Typesets the ⟨contents⟩ into a vertical box of height ⟨dimexpr⟩ and then includes this box in the current list for typesetting.

\vbox_to_zero:n \vbox_to_zero:n ⟨contents⟩
Typesets the ⟨contents⟩ into a vertical box of zero height and then includes this box in the current list for typesetting.

\vbox_set:Nn \vbox_set:cn \vbox_gset:Nn \vbox_gset:cn
Typesets the ⟨contents⟩ at natural height and then stores the result inside the ⟨box⟩.

\vbox_set_top:Nn \vbox_set_top:cn \vbox_gset_top:Nn \vbox_gset_top:cn
Typesets the ⟨contents⟩ at natural height and then stores the result inside the ⟨box⟩. The baseline of the box will be equal to that of the first item added to the box.

\vbox_set_to_ht:Nnn \vbox_set_to_ht:cn \vbox_gset_to_ht:Nnn \vbox_gset_to_ht:cn
Typesets the ⟨contents⟩ to the height given by the ⟨dimexpr⟩ and then stores the result inside the ⟨box⟩.

\vbox_set:Nw \vbox_set:cw \vbox_set_end:
\vbox_set:Nw ⟨box⟩ ⟨contents⟩ \vbox_set_end:
Typesets the ⟨contents⟩ at natural height and then stores the result inside the ⟨box⟩. In contrast to \vbox_set:Nn this function does not absorb the argument when finding the ⟨content⟩, and so can be used in circumstances where the ⟨content⟩ may not be a simple argument.

\vbox_set_split_to_ht:NNn \vbox_set_split_to_ht:cn \vbox_gset_split_to_ht:NNn \vbox_gset_split_to_ht:cn
Sets ⟨box1⟩ to contain material to the height given by the ⟨dimexpr⟩ by removing content from the top of ⟨box2⟩ (which must be a vertical box).

\TeXhackers~note: This is the \TeX~primitive \vsplit.
\vbox_unpack:N \vbox_unpack:N \vbox_pack:N \vbox_pack:N \vbox_pack:c
Unpacks the content of the vertical ⟨box⟩, retaining any stretching or shrinking applied when the ⟨box⟩ was set.

\TeXhacks{This is the \TeX primitive \unvcopy.}

\vbox_unpack_clear:N \vbox_unpack_clear:c
Unpacks the content of the vertical ⟨box⟩, retaining any stretching or shrinking applied when the ⟨box⟩ was set. The ⟨box⟩ is then cleared globally.

\TeXhacks{This is the \TeX primitive \unvbox.}

11 Primitive box conditionals

\if hbox:N \if hbox:N ⟨box⟩
 ⟨true code⟩
\else:
 ⟨false code⟩
\fi:
Tests is ⟨box⟩ is a horizontal box.

\TeXhacks{This is the \TeX primitive \ifhbox.}

\if vbox:N \if vbox:N ⟨box⟩
 ⟨true code⟩
\else:
 ⟨false code⟩
\fi:
Tests is ⟨box⟩ is a vertical box.

\TeXhacks{This is the \TeX primitive \ifvbox.}

\if_box_empty:N \if_box_empty:N ⟨box⟩
 ⟨true code⟩
\else:
 ⟨false code⟩
\fi:
Tests is ⟨box⟩ is an empty (void) box.

\TeXhacks{This is the \TeX primitive \ifvoid.}
Part XVII

The \texttt{l3coffins} package

Coffin code layer

The material in this module provides the low-level support system for coffins. For details about the design concept of a coffin, see the \texttt{xcoffins} module (in the \texttt{l3experimental} bundle).

1 Creating and initialising coffins

\begin{verbatim}
\coffin_new:N \coffin_new:C
New: 2011-08-17
\end{verbatim}

\coffin_new:N \coffin_new:C

Creates a new \texttt{(coffin)} or raises an error if the name is already taken. The declaration is global. The \texttt{(coffin)} will initially be empty.

\begin{verbatim}
\coffin_clear:N \coffin_clear:C
New: 2011-08-17
\end{verbatim}

\coffin_clear:N \coffin_clear:C

Clears the content of the \texttt{(coffin)} within the current \TeX{} group level.

\begin{verbatim}
\coffin_set_eq:NN \coffin_set_eq:cn
New: 2011-08-17
\end{verbatim}

\coffin_set_eq:NN \coffin_set_eq:cn

Sets both the content and poles of \texttt{(coffin\textsubscript{1})} equal to those of \texttt{(coffin\textsubscript{2})} within the current \TeX{} group level.

\begin{verbatim}
\coffin_if_exist_p:N \coffin_if_exist_p:c
\coffin_if_exist:N \coffin_if_exist_c
\coffin_if_exist_p:NTF \coffin_if_exist_p:NTF
\coffin_if_exist:N \coffin_if_exist_c
\coffin_if_exist_p:NTF \coffin_if_exist_p:NTF
New: 2012-06-20
\end{verbatim}

\coffin_if_exist_p:N \coffin_if_exist_p:c
\coffin_if_exist:N \coffin_if_exist_c
\coffin_if_exist_p:NTF \coffin_if_exist_p:NTF
\coffin_if_exist:N \coffin_if_exist_c
\coffin_if_exist_p:NTF \coffin_if_exist_p:NTF

Tests whether the \texttt{(coffin)} is currently defined.

2 Setting coffin content and poles

All coffin functions create and manipulate coffins locally within the current \TeX{} group level.

\begin{verbatim}
\hcoffin_set:Nn \hcoffin_set:cn
New: 2011-08-17
Updated: 2011-09-03
\end{verbatim}

\hcoffin_set:Nn \hcoffin_set:cn

Typesets the \texttt{(material)} in horizontal mode, storing the result in the \texttt{(coffin)}. The standard poles for the \texttt{(coffin)} are then set up based on the size of the typeset material.
\texttt{\hcoffin\_set:}\langle\texttt{coffin}\rangle\langle\texttt{material}\rangle\texttt{\hcoffin\_set\_end:}

Typesets the \langle\texttt{material}\rangle in horizontal mode, storing the result in the \langle\texttt{coffin}\rangle. The standard poles for the \langle\texttt{coffin}\rangle are then set up based on the size of the typeset material. These functions are useful for setting the entire contents of an environment in a coffin.

\texttt{\vcoffin\_set:}\langle\texttt{coffin}\rangle\{\langle\texttt{width}\rangle\}\{\langle\texttt{material}\rangle\}

Typesets the \langle\texttt{material}\rangle in vertical mode constrained to the given \langle\texttt{width}\rangle and stores the result in the \langle\texttt{coffin}\rangle. The standard poles for the \langle\texttt{coffin}\rangle are then set up based on the size of the typeset material.

\texttt{\vcoffin\_set:}\langle\texttt{coffin}\rangle\{\langle\texttt{width}\rangle\}\langle\texttt{material}\rangle\texttt{\vcoffin\_set\_end:}

Typesets the \langle\texttt{material}\rangle in vertical mode constrained to the given \langle\texttt{width}\rangle and stores the result in the \langle\texttt{coffin}\rangle. The standard poles for the \langle\texttt{coffin}\rangle are then set up based on the size of the typeset material. These functions are useful for setting the entire contents of an environment in a coffin.

\texttt{\coffin\_set\_horizontal\_pole:}\langle\texttt{coffin}\rangle\langle\texttt{pole}\rangle\{\langle\texttt{offset}\rangle\}

Sets the \langle\texttt{pole}\rangle to run horizontally through the \langle\texttt{coffin}\rangle. The \langle\texttt{pole}\rangle will be located at the \langle\texttt{offset}\rangle from the bottom edge of the bounding box of the \langle\texttt{coffin}\rangle. The \langle\texttt{offset}\rangle should be given as a dimension expression.

\texttt{\coffin\_set\_vertical\_pole:}\langle\texttt{coffin}\rangle\langle\texttt{pole}\rangle\{\langle\texttt{offset}\rangle\}

Sets the \langle\texttt{pole}\rangle to run vertically through the \langle\texttt{coffin}\rangle. The \langle\texttt{pole}\rangle will be located at the \langle\texttt{offset}\rangle from the left-hand edge of the bounding box of the \langle\texttt{coffin}\rangle. The \langle\texttt{offset}\rangle should be given as a dimension expression.
3 Joining and using coffins

This function attaches \(\text{coffin}_2\) to \(\text{coffin}_1\) such that the bounding box of \(\text{coffin}_1\) is not altered, i.e. \(\text{coffin}_2\) can protrude outside of the bounding box of the coffin. The alignment is carried out by first calculating \(\text{handle}_1\), the point of intersection of \(\text{coffin}_1\)-pole\(_1\) and \(\text{coffin}_1\)-pole\(_2\), and \(\text{handle}_2\), the point of intersection of \(\text{coffin}_2\)-pole\(_1\) and \(\text{coffin}_2\)-pole\(_2\). \(\text{coffin}_2\) is then attached to \(\text{coffin}_1\) such that the relationship between \(\text{handle}_1\) and \(\text{handle}_2\) is described by the \(x\)-offset and \(y\)-offset. The two offsets should be given as dimension expressions.

This function joins \(\text{coffin}_2\) to \(\text{coffin}_1\) such that the bounding box of \(\text{coffin}_1\) may expand. The new bounding box will cover the area containing the bounding boxes of the two original coffins. The alignment is carried out by first calculating \(\text{handle}_1\), the point of intersection of \(\text{coffin}_1\)-pole\(_1\) and \(\text{coffin}_1\)-pole\(_2\), and \(\text{handle}_2\), the point of intersection of \(\text{coffin}_2\)-pole\(_1\) and \(\text{coffin}_2\)-pole\(_2\). \(\text{coffin}_2\) is then attached to \(\text{coffin}_1\) such that the relationship between \(\text{handle}_1\) and \(\text{handle}_2\) is described by the \(x\)-offset and \(y\)-offset. The two offsets should be given as dimension expressions.

Typesetting is carried out by first calculating \(\text{handle}\), the point of intersection of \(\text{pole}_1\) and \(\text{pole}_2\). The coffin is then typeset in horizontal mode such that the relationship between the current reference point in the document and the \(\text{handle}\) is described by the \(x\)-offset and \(y\)-offset. The two offsets should be given as dimension expressions. Typesetting a coffin is therefore analogous to carrying out an alignment where the “parent” coffin is the current insertion point.

4 Measuring coffins

Calculates the depth (below the baseline) of the \(\text{coffin}\) in a form suitable for use in a \(\text{dimension expression}\).
\texttt{coffin\_ht:N} \texttt{coffin\_ht:c}
Calculates the height (above the baseline) of the \texttt{(coffin)} in a form suitable for use in a \texttt{(dimension expression)}.

\texttt{coffin\_wd:N} \texttt{coffin\_wd:c}
Calculates the width of the \texttt{(coffin)} in a form suitable for use in a \texttt{(dimension expression)}.

5 Coffin diagnostics

\texttt{coffin\_display\_handles:Nn} \texttt{coffin\_display\_handles:cn}
This function first calculates the intersections between all of the \texttt{(poles)} of the \texttt{(coffin)} to give a set of \texttt{(handles)}. It then prints the \texttt{(coffin)} at the current location in the source, with the position of the \texttt{(handles)} marked on the coffin. The \texttt{(handles)} will be labelled as part of this process: the locations of the \texttt{(handles)} and the labels are both printed in the \texttt{(color)} specified.

\texttt{coffin\_mark\_handle:Nnnn} \texttt{coffin\_mark\_handle:cn}
This function first calculates the \texttt{(handle)} for the \texttt{(coffin)} as defined by the intersection of \texttt{(pole1)} and \texttt{(pole2)}. It then marks the position of the \texttt{(handle)} on the \texttt{(coffin)}. The \texttt{(handle)} will be labelled as part of this process: the location of the \texttt{(handle)} and the label are both printed in the \texttt{(color)} specified.

\texttt{coffin\_show\_structure:N} \texttt{coffin\_show\_structure:c}
This function shows the structural information about the \texttt{(coffin)} in the terminal. The width, height and depth of the typeset material are given, along with the location of all of the poles of the coffin.
Notice that the poles of a coffin are defined by four values: the \texttt{x} and \texttt{y} co-ordinates of a point that the pole passes through and the \texttt{x-} and \texttt{y}-components of a vector denoting the direction of the pole. It is the ratio between the later, rather than the absolute values, which determines the direction of the pole.

5.1 Constants and variables

\texttt{c\_empty\_coffin}
A permanently empty coffin.

\texttt{l\_tmpa\_coffin} \texttt{l\_tmpb\_coffin}
Scratch coffins for local assignment. These are never used by the kernel code, and so are safe for use with any \LaTeX\-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

Updated: 2012-09-09

Updated: 2011-09-02
Part XVIII

The l3color package

Color support

This module provides support for color in \LaTeXe. At present, the material here is mainly intended to support a small number of low-level requirements in other l3kernel modules.

1 Color in boxes

Controlling the color of text in boxes requires a small number of control functions, so that the boxed material uses the color at the point where it is set, rather than where it is used.

\begin{verbatim}
\color_group_begin: ...
\color_group_end:
\end{verbatim}

New: 2011-09-03

Creates a color group: one used to “trap” color settings.

\begin{verbatim}
\color_ensure_current:
\end{verbatim}

New: 2011-09-03

Ensures that material inside a box will use the foreground color at the point where the box is set, rather than that in force when the box is used. This function should usually be used within a \color_group_begin: ...\color_group_end: group.
Part XIX

The l3msg package

Messages

Messages need to be passed to the user by modules, either when errors occur or to indicate how the code is proceeding. The l3msg module provides a consistent method for doing this (as opposed to writing directly to the terminal or log).

The system used by l3msg to create messages divides the process into two distinct parts. Named messages are created in the first part of the process; at this stage, no decision is made about the type of output that the message will produce. The second part of the process is actually producing a message. At this stage a choice of message class has to be made, for example error, warning or info.

By separating out the creation and use of messages, several benefits are available. First, the messages can be altered later without needing details of where they are used in the code. This makes it possible to alter the language used, the detail level and so on. Secondly, the output which results from a given message can be altered. This can be done on a message class, module or message name basis. In this way, message behaviour can be altered and messages can be entirely suppressed.

1 Creating new messages

All messages have to be created before they can be used. The text of messages will automatically be wrapped to the length available in the console. As a result, formatting is only needed where it will help to show meaning. In particular, \ may be used to force a new line and \ forces an explicit space. Additionally, \, \#, \}, \% and \ can be used to produce the corresponding character.

Messages may be subdivided by one level using the / character. This is used within the message filtering system to allow for example the \LaTeX{} kernel messages to belong to the module \LaTeX{} while still being filterable at a more granular level. Thus for example

\msg_new:nnnn { mymodule } { submodule / message } ...

will allow only those messages from the submodule to be filtered out.

\msg_new:nnnn \msg_new:nnn

Updated: 2011-08-16

\msg_new:nnnn {⟨module⟩} {⟨message⟩} {⟨text⟩} {⟨more text⟩}

Creates a ⟨message⟩ for a given ⟨module⟩. The message will be defined to first give ⟨text⟩ and then ⟨more text⟩ if the user requests it. If no ⟨more text⟩ is available then a standard text is given instead. Within ⟨text⟩ and ⟨more text⟩ four parameters (#1 to #4) can be used; these will be supplied at the time the message is used. An error will be raised if the ⟨message⟩ already exists.
Sets up the text for a ⟨message⟩ for a given ⟨module⟩. The message will be defined to first give ⟨text⟩ and then ⟨more text⟩ if the user requests it. If no ⟨more text⟩ is available then a standard text is given instead. Within ⟨text⟩ and ⟨more text⟩ four parameters (#1 to #4) can be used: these will be supplied at the time the message is used.

Tests whether the ⟨message⟩ for the ⟨module⟩ is currently defined.

### 2 Contextual information for messages

*\texttt{\msg_line_context}: ~\star\texttt* *\texttt{\msg_line_context}:*

Prints the current line number when a message is given, and thus suitable for giving context to messages. The number itself is proceeded by the text on line.

*\texttt{\msg_line_number}: ~\star\texttt* *\texttt{\msg_line_number}:

Prints the current line number when a message is given.

*\texttt{\msg_fatal_text:n}: ~\star\texttt* *\texttt{\msg_fatal_text:n}:

Produces the standard text

\texttt{Fatal \langle\texttt{module}\rangle \textit{error}}

This function can be redefined to alter the language in which the message is given, using #1 as the name of the ⟨module⟩ to be included.

*\texttt{\msg_critical_text:n}: ~\star\texttt* *\texttt{\msg_critical_text:n}:

Produces the standard text

\texttt{Critical \langle\texttt{module}\rangle \textit{error}}

This function can be redefined to alter the language in which the message is given, using #1 as the name of the ⟨module⟩ to be included.

*\texttt{\msg_error_text:n}: ~\star\texttt* *\texttt{\msg_error_text:n}:

Produces the standard text

\texttt{\langle\texttt{module}\rangle \textit{error}}

This function can be redefined to alter the language in which the message is given, using #1 as the name of the ⟨module⟩ to be included.
\texttt{\msg.warning.text:n \*} \begin{verbatim}
\{\langle\text{module}\rangle\}
\end{verbatim}

Produces the standard text 

\begin{verbatim}
\langle\text{module}\rangle\text{ warning}
\end{verbatim}

This function can be redefined to alter the language in which the message is given, using \#1 as the name of the \langle\text{module}\rangle to be included.

\texttt{\msg.info.text:n \*} \begin{verbatim}
\{\langle\text{module}\rangle\}
\end{verbatim}

Produces the standard text: 

\begin{verbatim}
\langle\text{module}\rangle\text{ info}
\end{verbatim}

This function can be redefined to alter the language in which the message is given, using \#1 as the name of the \langle\text{module}\rangle to be included.

\texttt{\msg.see.documentation.text:n \*} \begin{verbatim}
\{\langle\text{module}\rangle\}
\end{verbatim}

Produces the standard text

\begin{verbatim}
See the \langle\text{module}\rangle documentation for further information.
\end{verbatim}

This function can be redefined to alter the language in which the message is given, using \#1 as the name of the \langle\text{module}\rangle to be included.

\section{Issuing messages}

Messages behave differently depending on the message class. In all cases, the message may be issued supplying 0 to 4 arguments. If the number of arguments supplied here does not match the number in the definition of the message, extra arguments will be ignored, or empty arguments added (of course the sense of the message may be impaired). The four arguments will be converted to strings before being added to the message text: the x-type variants should be used to expand material.

\texttt{\msg.fatal:nnnnnn \*} \begin{verbatim}
\{\langle\text{module}\rangle\}\{\langle\text{message}\rangle\}\{\langle\text{arg one}\rangle\}\{\langle\text{arg two}\rangle\}\{\langle\text{arg three}\rangle\}\{\langle\text{arg four}\rangle\}
\end{verbatim}

Issues \langle\text{module}\rangle error \langle\text{message}\rangle, passing \langle\text{arg one}\rangle to \langle\text{arg four}\rangle to the text-creating functions. After issuing a fatal error the \TeX{} run will halt.
\texttt{\textbackslash msg\_critical:nnnnnn}\{\textbackslash module\}\{\textbackslash message\}\{\textbackslash arg\ one\}\{\textbackslash arg\ two\}\{\textbackslash arg\ three\}\{\textbackslash arg\ four\}\}

Issues \textit{(module)} error \textit{(message)}, passing \textit{(arg one)} to \textit{(arg four)} to the text-creating functions. After issuing a critical error, \TeX{} will stop reading the current input file. This may halt the \TeX{} run (if the current file is the main file) or may abort reading a sub-file.

\textbf{\TeX{}hackers note:} The \TeX{} \texttt{\textbackslash endinput} primitive is used to exit the file. In particular, the rest of the current line remains in the input stream.

\texttt{\textbackslash msg\_error:nnnnnn}\{\textbackslash module\}\{\textbackslash message\}\{\textbackslash arg\ one\}\{\textbackslash arg\ two\}\{\textbackslash arg\ three\}\{\textbackslash arg\ four\}\}

Issues \textit{(module)} error \textit{(message)}, passing \textit{(arg one)} to \textit{(arg four)} to the text-creating functions. The error will interrupt processing and issue the text at the terminal. After user input, the run will continue.

\texttt{\textbackslash msg\_warning:nnnnnn}\{\textbackslash module\}\{\textbackslash message\}\{\textbackslash arg\ one\}\{\textbackslash arg\ two\}\{\textbackslash arg\ three\}\{\textbackslash arg\ four\}\}

Issues \textit{(module)} warning \textit{(message)}, passing \textit{(arg one)} to \textit{(arg four)} to the text-creating functions. The warning text will be added to the log file and the terminal, but the \TeX{} run will not be interrupted.

\texttt{\textbackslash msg\_info:nnnnnn}\{\textbackslash module\}\{\textbackslash message\}\{\textbackslash arg\ one\}\{\textbackslash arg\ two\}\{\textbackslash arg\ three\}\{\textbackslash arg\ four\}\}

Issues \textit{(module)} information \textit{(message)}, passing \textit{(arg one)} to \textit{(arg four)} to the text-creating functions. The information text will be added to the log file.
4 Redirecting messages

Each message has a “name”, which can be used to alter the behaviour of the message when it is given. Thus we might have

\msg_new:nnnn { module } { my-message } { Some-text } { Some-more-text }

to define a message, with

\msg_error:nn { module } { my-message }

when it is used. With no filtering, this will raise an error. However, we could alter the behaviour with

\msg_redirect_class:nn { error } { warning }

to turn all errors into warnings, or with

\msg_redirect_module:nnn { module } { error } { warning }

to alter only messages from that module, or even

\msg_redirect_name:nnn { module } { my-message } { warning }

Issues (module) information (message), passing (arg one) to (arg four) to the text-creating functions. The information text will be added to the log file: the output is briefer than \msg_info:nnnnn.

Does nothing: used as a message class to prevent any output at all (see the discussion of message redirection).
to target just one message. Redirection applies first to individual messages, then to messages from one module and finally to messages of one class. Thus it is possible to select out an individual message for special treatment even if the entire class is already redirected.

Multiple redirections are possible. Redirections can be cancelled by providing an empty argument for the target class. Redirection to a missing class will raise errors immediately. Infinite loops are prevented by eliminating the redirection starting from the target of the redirection that caused the loop to appear. Namely, if redirections are requested as $A \rightarrow B$, $B \rightarrow C$ and $C \rightarrow A$ in this order, then the $A \rightarrow B$ redirection is cancelled.

\texttt{\textbackslash{}msg\_redirect\_class:nn} \{\langle \text{class one} \rangle\} \{\langle \text{class two} \rangle\}

Changes the behaviour of messages of \langle class one \rangle so that they are processed using the code for those of \langle class two \rangle.

\texttt{\textbackslash{}msg\_redirect\_module:nnn} \{\langle \text{module} \rangle\} \{\langle \text{class one} \rangle\} \{\langle \text{class two} \rangle\}

Redirects message of \langle class one \rangle for \langle module \rangle to act as though they were from \langle class two \rangle. Messages of \langle class one \rangle from sources other than \langle module \rangle are not affected by this redirection. This function can be used to make some messages “silent” by default. For example, all of the \texttt{warning} messages of \langle module \rangle could be turned off with:

\texttt{\textbackslash{}msg\_redirect\_module:nnn \{ module \} \{ warning \} \{ none \}}

\texttt{\textbackslash{}msg\_redirect\_name:nnn} \{\langle \text{module} \rangle\} \{\langle \text{message} \rangle\} \{\langle \text{class} \rangle\}

Redirects a specific \langle message \rangle from a specific \langle module \rangle to act as a member of \langle class \rangle of messages. No further redirection is performed. This function can be used to make a selected message “silent” without changing global parameters:

\texttt{\textbackslash{}msg\_redirect\_name:nnn \{ module \} \{ annoying-message \} \{ none \}}

5 Low-level message functions

The lower-level message functions should usually be accessed from the higher-level system. However, there are occasions where direct access to these functions is desirable.
\msg_interrupt:nn
Interrupts the \TeX{} run, issuing a formatted message comprising \{first line\} and \{text\} laid out in the format

\begin{verbatim}
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! \langle first line\rangle
! \langle text\rangle
\end{verbatim}

where the \textit{\langle text\rangle} will be wrapped to fit within the current line length. The user may then request more information, at which stage the \textit{\langle extra text\rangle} will be shown in the terminal in the format

\begin{verbatim}
|''''''''''''''''''''''''''''''''''''''''''''''''''''''
| \langle extra text\rangle
\end{verbatim}

where the \textit{\langle extra text\rangle} will be wrapped within the current line length. Wrapping of both \textit{\langle text\rangle} and \textit{\langle more text\rangle} takes place using \texttt{\iow_wrap:nnnN}; the documentation for the latter should be consulted for full details.

\msg_log:n
\texttt{\msg_log:n \{text\}}

Writes to the log file with the \textit{\langle text\rangle} laid out in the format

\begin{verbatim}
..............................
. \langle text\rangle
..............................
\end{verbatim}

where the \textit{\langle text\rangle} will be wrapped to fit within the current line length. Wrapping takes place using \texttt{\iow_wrap:nnnN}; the documentation for the latter should be consulted for full details.

\msg_term:n
\texttt{\msg_term:n \{text\}}

Writes to the terminal and log file with the \textit{\langle text\rangle} laid out in the format

\begin{verbatim}
*************************************************
* \langle text\rangle
*************************************************
\end{verbatim}

where the \textit{\langle text\rangle} will be wrapped to fit within the current line length. Wrapping takes place using \texttt{\iow_wrap:nnnN}; the documentation for the latter should be consulted for full details.
6 Kernel-specific functions

Messages from \TeX{}3 itself are handled by the general message system, but have their own functions. This allows some text to be pre-defined, and also ensures that serious errors can be handled properly.

\begin{verbatim}
\__msg_kernel_new:nnnn {⟨module⟩} {⟨message⟩} {⟨text⟩} {⟨more text⟩}
Creates a kernel ⟨message⟩ for a given ⟨module⟩. The message will be defined to first give ⟨text⟩ and then ⟨more text⟩ if the user requests it. If no ⟨more text⟩ is available then a standard text is given instead. Within ⟨text⟩ and ⟨more text⟩ four parameters ( #1 to #4 ) can be used: these will be supplied and expanded at the time the message is used. An error will be raised if the ⟨message⟩ already exists.
\end{verbatim}

\begin{verbatim}
\__msg_kernel_set:nnnn {⟨module⟩} {⟨message⟩} {⟨text⟩} {⟨more text⟩}
Sets up the text for a kernel ⟨message⟩ for a given ⟨module⟩. The message will be defined to first give ⟨text⟩ and then ⟨more text⟩ if the user requests it. If no ⟨more text⟩ is available then a standard text is given instead. Within ⟨text⟩ and ⟨more text⟩ four parameters ( #1 to #4 ) can be used: these will be supplied and expanded at the time the message is used.
\end{verbatim}

\begin{verbatim}
\__msg_kernel_fatal:nnnnn {⟨module⟩} {⟨message⟩} {⟨arg one⟩} {⟨arg two⟩} {⟨arg three⟩} {⟨arg four⟩}
Issues kernel ⟨module⟩ error ⟨message⟩, passing ⟨arg one⟩ to ⟨arg four⟩ to the text-creating functions. After issuing a fatal error the \TeX{}X run will halt. Cannot be redirected.
\end{verbatim}

\begin{verbatim}
\__msg_kernel_error:nnnnn {⟨module⟩} {⟨message⟩} {⟨arg one⟩} {⟨arg two⟩} {⟨arg three⟩} {⟨arg four⟩}
Issues kernel ⟨module⟩ error ⟨message⟩, passing ⟨arg one⟩ to ⟨arg four⟩ to the text-creating functions. The error will stop processing and issue the text at the terminal. After user input, the run will continue. Cannot be redirected.
\end{verbatim}
Issues kernel ⟨module⟩ warning ⟨message⟩, passing ⟨arg one⟩ to ⟨arg four⟩ to the text-creating functions. The warning text will be added to the log file, but the TeX run will not be interrupted.

Issues kernel ⟨module⟩ information ⟨message⟩, passing ⟨arg one⟩ to ⟨arg four⟩ to the text-creating functions. The information text will be added to the log file.

7 Expandable errors

In a few places, the \LaTeX3 kernel needs to produce errors in an expansion only context. This must be handled internally very differently from normal error messages, as none of the tools to print to the terminal or the log file are expandable. However, the interface is similar, with the important caveat that the message text and arguments are not expanded, and messages should be very short.

Issues an error, passing ⟨arg one⟩ to ⟨arg four⟩ to the text-creating functions. The resulting string must be shorter than a line, otherwise it will be cropped.
Issues an “Undefined error” message from TeX itself, and prints the \textit{error message}. The \textit{error message} must be short: it is cropped at the end of one line.

\textbf{TeXhackers note:} This function expands to an empty token list after two steps. Tokens inserted in response to TeX’s prompt are read with the current category code setting, and inserted just after the place where the error message was issued.

## 8 Internal l3msg functions

The following functions are used in several kernel modules.

\begin{description}
\item \texttt{\_msg\_term:nnnnnn} \{\texttt{module}\} \{\texttt{message}\} \{\texttt{arg one}\} \{\texttt{arg two}\} \{\texttt{arg three}\} \{\texttt{arg four}\}
\end{description}

Prints the \textit{message} from \textit{module} in the terminal without formatting. Used in messages which print complex variable contents completely.

\begin{description}
\item \texttt{\_msg\_show\_variable:Nnn} \{\texttt{variable}\} \{\texttt{type}\} \{\texttt{formatted content}\}
\end{description}

Displays the \textit{formatted content} of the \textit{variable} of \textit{type} in the terminal. The \textit{formatted content} will be processed as the first argument in a call to \texttt{\_io\_wrap:nnnn}, hence \textbackslash, \textbackslash, \textbackslash, and other formatting sequences can be used. Once expanded and processed, the \textit{formatted content} must either be empty or contain \texttt{>}; everything until the first \texttt{>} will be removed.

\begin{description}
\item \texttt{\_msg\_show\_variable:n} \{\texttt{formatted text}\}
\end{description}

Shows the \textit{formatted text} on the terminal. After expansion, unless it is empty, the \textit{formatted text} must contain \texttt{>}, and the part of \textit{formatted text} before the first \texttt{>} is removed. Failure to do so causes low-level TeX errors.

Auxiliary functions used within the argument of \texttt{\_msg\_show\_variable:Nnn} to format variable items correctly for display. The \texttt{\_msg\_show\_item:n} version is used for simple lists, the \texttt{\_msg\_show\_item:nn} and \texttt{\_msg\_show\_item\_unbraced:nn} versions for key-value like data structures.
The text

*This is a coding error.*

used by kernel functions when erroneous programming input is encountered.
Part XX

The l3keys package

Key–value interfaces

The key–value method is a popular system for creating large numbers of settings for controlling function or package behaviour. The system normally results in input of the form

\MyModuleSetup{
  key-one = value one,
  key-two = value two
}

or

\MyModuleMacro{
  key-one = value one,
  key-two = value two
}{argument}

for the user.

The high level functions here are intended as a method to create key–value controls. Keys are themselves created using a key–value interface, minimising the number of functions and arguments required. Each key is created by setting one or more properties of the key:

\keys_define:nn { mymodule }
{
  key-one .code:n = code including parameter #1,
  key-two .tl_set:N = \l_mymodule_store_tl
}

These values can then be set as with other key–value approaches:

\keys_set:nn { mymodule }
{
  key-one = value one,
  key-two = value two
}

At a document level, \keys_set:nn will be used within a document function, for example

\DeclareDocumentCommand \MyModuleSetup { m }
{ \keys_set:nn { mymodule } { #1 } }
\DeclareDocumentCommand \MyModuleMacro { o m }
{
Key names may contain any tokens, as they are handled internally using \tl_to_str:n. As will be discussed in section 2, it is suggested that the character / is reserved for sub-division of keys into logical groups. Functions and variables are not expanded when creating key names, and so
\tl_set:Nn \l_mymodule_tmp_tl { key }
\keys_define:nn { mymodule }
{ 
  \l_mymodule_tmp_tl .code:n = code
}
will create a key called \l_mymodule_tmp_tl, and not one called key.

1 Creating keys

\keys_define:nn \keys_define:nn {⟨module⟩} {⟨keyval list⟩}
\keys_define:nn {langle module angle} {langle keyval list angle}
Parses the ⟨keyval list⟩ and defines the keys listed there for ⟨module⟩. The ⟨module⟩ name should be a text value, but there are no restrictions on the nature of the text. In practice the ⟨module⟩ should be chosen to be unique to the module in question (unless deliberately adding keys to an existing module).

The ⟨keyval list⟩ should consist of one or more key names along with an associated key property. The properties of a key determine how it acts. The individual properties are described in the following text; a typical use of \keys_define:nn might read
\keys_define:nn { mymodule }
{
  keyname .code:n = Some-code-using-#1,
  keyname .value_required:
}
where the properties of the key begin from the . after the key name.

The various properties available take either no arguments at all, or require one or more arguments. This is indicated in the name of the property using an argument specification. In the following discussion, each property is illustrated attached to an arbitrary ⟨key⟩, which when used may be supplied with a ⟨value⟩. All key definitions are local.
(key).bool_set:N = {boolean}
Defines (key) to set (boolean) to (value) (which must be either true or false). If the variable does not exist, it will be created globally at the point that the key is set up.

(key).bool_set_inverse:N = {boolean}
Defines (key) to set (boolean) to the logical inverse of (value) (which must be either true or false). If the (boolean) does not exist, it will be created globally at the point that the key is set up.

(key).choice:
Sets (key) to act as a choice key. Each valid choice for (key) must then be created, as discussed in section 3.

(key).choices:nn = {choices} (code)
Sets (key) to act as a choice key, and defines a series of choices which are implemented using the (code). Inside (code), \l_keys_choice_tl will be the name of the choice made, and \l_keys_choice_int will be the position of the choice in the list of choices (indexed from 1). Choices are discussed in detail in section 3.

(key).clist_set:N = {comma list variable}
Defines (key) to set (comma list variable) to (value). Spaces around commas and empty items will be stripped. If the variable does not exist, it will be created globally at the point that the key is set up.

(key).code:n = {(code)}
Stores the (code) for execution when (key) is used. The (code) can include one parameter (#1), which will be the (value) given for the (key). The x-type variant will expand (code) at the point where the (key) is created.
\keys_define:nn { mymodule }
{  
  key .code:n = Hello~#1,  
  key .default:n = World  
}
\keys_set:nn { mymodule }
{  
  key = Fred, % Prints 'Hello Fred'  
  key, % Prints 'Hello World'  
  key = , % Prints 'Hello '  
}

\keys_set:nn { mymodule }
{  
  key .dim_set:N = <dimension>  
}
\keys_set:nn { mymodule }
{  
  key .fp_set:N = <floating point>  
}
\keys_set:nn { mymodule }
{  
  key .groups:n = { <groups> }  
}
\keys_set:nn { mymodule }
{  
  key .initial:n = <value>  
}
\keys_set:nn { mymodule }
{  
  key .int_set:N = <integer>  
}
〈key〉.meta:n = {〈keyval list〉}

Makes 〈key〉 a meta-key, which will set 〈keyval list〉 in one go. If 〈key〉 is given with a value at the time the key is used, then the value will be passed through to the subsidiary 〈keys〉 for processing (as #1).

〈key〉.meta:nn = {〈path〉} {〈keyval list〉}

Makes 〈key〉 a meta-key, which will set 〈keyval list〉 in one go using the 〈path〉 in place of the current one. If 〈key〉 is given with a value at the time the key is used, then the value will be passed through to the subsidiary 〈keys〉 for processing (as #1).

〈key〉.multichoice:

Sets 〈key〉 to act as a multiple choice key. Each valid choice for 〈key〉 must then be created, as discussed in section 3.

〈key〉.multichoices:nn {〈choices〉} {〈code〉}

Sets 〈key〉 to act as a multiple choice key, and defines a series 〈choices〉 which are implemented using the 〈code〉. Inside 〈code〉, \l_keys_choice_tl will be the name of the choice made, and \l_keys_choice_int will be the position of the choice in the list of 〈choices〉 (indexed from 1). Choices are discussed in detail in section 3.

〈key〉.skip_set:N = 〈skip〉

Defines 〈key〉 to set 〈skip〉 to 〈value〉 (which must be a skip expression). If the variable does not exist, it will be created globally at the point that the key is set up.

〈key〉.tl_set:N = 〈token list variable〉

Defines 〈key〉 to set 〈token list variable〉 to 〈value〉. If the variable does not exist, it will be created globally at the point that the key is set up.

〈key〉.tl_set_x:N = 〈token list variable〉

Defines 〈key〉 to set 〈token list variable〉 to 〈value〉, which will be subjected to an x-type expansion (i.e. using \tl_set:Nx). If the variable does not exist, it will be created globally at the point that the key is set up.

〈key〉.value_forbidden:

Specifies that 〈key〉 cannot receive a 〈value〉 when used. If a 〈value〉 is given then an error will be issued.

〈key〉.value_required:

Specifies that 〈key〉 must receive a 〈value〉 when used. If a 〈value〉 is not given then an error will be issued.
2 Sub-dividing keys

When creating large numbers of keys, it may be desirable to divide them into several sub-groups for a given module. This can be achieved either by adding a sub-division to the module name:

\keys_define:nn { module / subgroup }
{ key .code:n = code }

or to the key name:

\keys_define:nn { mymodule }
{ subgroup / key .code:n = code }

As illustrated, the best choice of token for sub-dividing keys in this way is /. This is because of the method that is used to represent keys internally. Both of the above code fragments set the same key, which has full name module/subgroup/key. As will be illustrated in the next section, this subdivision is particularly relevant to making multiple choices.

3 Choice and multiple choice keys

The l3keys system supports two types of choice key, in which a series of pre-defined input values are linked to varying implementations. Choice keys are usually created so that the various values are mutually-exclusive: only one can apply at any one time. “Multiple” choice keys are also supported: these allow a selection of values to be chosen at the same time.

Mutually-exclusive choices are created by setting the .choice: property:

\keys_define:nn { mymodule }
{ key .choice: }

For keys which are set up as choices, the valid choices are generated by creating sub-keys of the choice key. This can be carried out in two ways.

In many cases, choices execute similar code which is dependant only on the name of the choice or the position of the choice in the list of all possibilities. Here, the keys can share the same code, and can be rapidly created using the .choices:nn property.

\keys_define:nn { mymodule }
{ key .choices:nn =
{ choice-a, choice-b, choice-c }
{ You-gave-choice-\tl_use:N \l_keys_choice_tl, which-is-in-position-\int_use:N \l_keys_choice_int \c_space_tl in-the-list. }
}
The index `\l_keys_choice_int` in the list of choices starts at 1.

Inside the code block for a choice generated using `.choices:nn`, the variables `\l_keys_choice_tl` and `\l_keys_choice_int` are available to indicate the name of the current choice, and its position in the comma list. The position is indexed from 1. Note that, as with standard key code generated using `.code:n`, the value passed to the key (i.e. the choice name) is also available as `#1`.

On the other hand, it is sometimes useful to create choices which use entirely different code from one another. This can be achieved by setting the `.choice:` property of a key, then manually defining sub-keys.

```
\keys_define:nn { mymodule }
{  
  key .choice:,  
  key / choice-a .code:n = code-a,  
  key / choice-b .code:n = code-b,  
  key / choice-c .code:n = code-c,  
}
```

It is possible to mix the two methods, but manually-created choices should *not* use `\l_keys_choice_tl` or `\l_keys_choice_int`. These variables do not have defined behaviour when used outside of code created using `.choices:nn` (i.e. anything might happen).

It is possible to allow choice keys to take values which have not previously been defined by adding code for the special `unknown` choice. The general behavior of the `unknown` key is described in Section 5. A typical example in the case of a choice would be to issue a custom error message:

```
\keys_define:nn { mymodule }
{  
  key .choice:,  
  key / choice-a .code:n = code-a,  
  key / choice-b .code:n = code-b,  
  key / choice-c .code:n = code-c,  
  key / unknown .code:n =  
    \msg_error:nnxxx { mymodule } { unknown-choice }  
    { key } % Name of choice key  
    { choice-a , choice-b , choice-c } % Valid choices  
    { \exp_not:n {#1} } % Invalid choice given  
  }
```

Multiple choices are created in a very similar manner to mutually-exclusive choices, using the properties `.multchoicexx` and `.multicoxxes:nn`. As with mutually exclusive choices, multiple choices are define as sub-keys. Thus both
\keys_define:nn { mymodule }
{
  key .multichoices:nn =
  { choice-a, choice-b, choice-c }
  {
    You-gave-choice-\tl_use:N \l_keys_choice_tl',-
    which-is-in-position-
    \int_use:N \l_keys_choice_int \c_space_tl
    in-the-list.
  }
}

and

\keys_define:nn { mymodule }
{
  key .multichoice:,
  key / choice-a .code:n = code-a,
  key / choice-b .code:n = code-b,
  key / choice-c .code:n = code-c,
}

are valid.

When a multiple choice key is set

\keys_set:nn { mymodule }
{
  key = { a , b , c } % 'key' defined as a multiple choice
}

each choice is applied in turn, equivalent to a cлист mapping or to applying each value individually:

\keys_set:nn { mymodule }
{
  key = a ,
  key = b ,
  key = c ,
}

Thus each separate choice will have passed to it the \l_keys_choice_tl and \l_keys_choice_int in exactly the same way as described for .choices:nn.

4 Setting keys

\keys_set:nn \keys_set:nn {⟨module⟩} {⟨keyval list⟩}
\keys_set:nn {⟨module⟩} {⟨keyval list⟩}

Parses the ⟨keyval list⟩, and sets those keys which are defined for ⟨module⟩. The behaviour on finding an unknown key can be set by defining a special unknown key: this will be illustrated later.
For each key processed, information of the full path of the key, the name of the key and the value of the key is available within three token list variables. These may be used within the code of the key.

The value is everything after the =, which may be empty if no value was given. This is stored in \l_keys_value_tl, and is not processed in any way by \keys_set:nn.

The path of the key is a “full” description of the key, and is unique for each key. It consists of the module and full key name, thus for example

\keys_set:nn { mymodule } { key-a = some-value }

has path mymodule/key-a while

\keys_set:nn { mymodule } { subset / key-a = some-value }

has path mymodule/subset/key-a. This information is stored in \l_keys_path_tl, and will have been processed by \tl_to_str:n.

The name of the key is the part of the path after the last /, and thus is not unique. In the preceding examples, both keys have name key-a despite having different paths. This information is stored in \l_keys_key_tl, and will have been processed by \tl_to_str:n.

## 5 Handling of unknown keys

If a key has not previously been defined (is unknown), \keys_set:nn will look for a special unknown key for the same module, and if this is not defined raises an error indicating that the key name was unknown. This mechanism can be used for example to issue custom error texts.

\keys_define:nn { mymodule }
{  
unknown.code:n =  
You-tried-to-set-key-’\l_keys_key_tl’-to-’#1’.  
}
In some cases, the desired behavior is to simply ignore unknown keys, collecting up information on these for later processing. The \texttt{\keys_set_known:nnN} function parses the \texttt{(keyval list)}, and sets those keys which are defined for \texttt{(module)}. Any keys which are unknown are not processed further by the parser. The key-value pairs for each unknown key name will be stored in the \texttt{(tl)} in a comma-separated form \textit{(i.e.} an edited version of the \texttt{(keyval list)}). The \texttt{\keys_set_known:nn} version skips this stage.

Use of \texttt{\keys_set_known:nnN} can be nested, with the correct residual \texttt{(keyval list)} returned at each stage.

6 Selective key setting

In some cases it may be useful to be able to select only some keys for setting, even though these keys have the same path. For example, with a set of keys defined using

\begin{verbatim}
\keys define:nn { mymodule }
    { 
        key-one .code:n = \\my_func:n {#1} ,
        key-two .tl_set:N = \l_my_a_tl ,
        key-three .tl_set:N = \l_my_b_tl ,
        key-four .fp_set:N = \l_my_a_fp ,
    }
\end{verbatim}

the use of \texttt{\keys_set:nn} will attempt to set all four keys. However, in some contexts it may only be sensible to set some keys, or to control the order of setting. To do this, keys may be assigned to \texttt{groups}: arbitrary sets which are independent of the key tree. Thus modifying the example to read

\begin{verbatim}
\keys define:nn { mymodule }
    { 
        key-one .code:n = \\my_func:n {#1} ,
        key-one .groups:n = \first ,
        key-two .tl_set:N = \l_my_a_tl ,
        key-two .groups:n = \first ,
        key-three .tl_set:N = \l_my_b_tl ,
        key-three .groups:n = \second ,
        key-four .fp_set:N = \l_my_a_fp ,
    }
\end{verbatim}

will assign \texttt{key-one} and \texttt{key-two} to group \texttt{first}, \texttt{key-three} to group \texttt{second}, while \texttt{key-four} is not assigned to a group.
Selective key setting may be achieved either by selecting one or more groups to be made “active”, or by marking one or more groups to be ignored in key setting.

\keys_set_filter:nnnN \keys_set_filter:nnnN \keys_set_filter:nnnN \keys_set_groups:nnn \keys_set_groups:nnn

New: 2013-07-14
Updated: 2014-04-27

Actives key filtering in an “opt-out” sense: keys assigned to any of the \langle groups \rangle specified will be ignored. The \langle groups \rangle are given as a comma-separated list. Unknown keys are not assigned to any group and will thus always be set. The key–value pairs for each key which is filtered out will be stored in the \langle tl \rangle in a comma-separated form (i.e. an edited version of the \langle keyval list \rangle). The \keys_set_filter:nnn version skips this stage.

Use of \keys_set_filter:nnnN can be nested, with the correct residual \langle keyval list \rangle returned at each stage.

\keys_set_groups:nnn \keys_set_groups:nnn

New: 2013-07-14

Actives key filtering in an “opt-in” sense: only keys assigned to one or more of the \langle groups \rangle specified will be set. The \langle groups \rangle are given as a comma-separated list. Unknown keys are not assigned to any group and will thus never be set.

7 Utility functions for keys

\keys_if_exist_p:nn \keys_if_exist_p:nn \keys_if_exist:nnTF \keys_if_exist:nnTF

Tests if the \langle key \rangle exists for \langle module \rangle, i.e. if any code has been defined for \langle key \rangle.

\keys_if_choice_exist_p:nnn \keys_if_choice_exist_p:nnn \keys_if_choice_exist:nnnTF \keys_if_choice_exist:nnnTF

Tests if the \langle choice \rangle is defined for the \langle key \rangle within the \langle module \rangle, i.e. if any code has been defined for \langle key \rangle/\langle choice \rangle. The test is false if the \langle key \rangle itself is not defined.

\keys_show:nn \keys_show:nn

Shows the function which is used to actually implement a \langle key \rangle for a \langle module \rangle.
8 Low-level interface for parsing key–val lists

To re-cap from earlier, a key–value list is input of the form

\[
\text{KeyOne} = \text{ValueOne} , \\
\text{KeyTwo} = \text{ValueTwo} , \\
\text{KeyThree}
\]

where each key–value pair is separated by a comma from the rest of the list, and each key–value pair does not necessarily contain an equals sign or a value! Processing this type of input correctly requires a number of careful steps, to correctly account for braces, spaces and the category codes of separators.

While the functions described earlier are used as a high-level interface for processing such input, in special circumstances you may wish to use a lower-level approach. The low-level parsing system converts a \((\text{key–value list})\) into \((\text{keys})\) and associated \((\text{values})\). After the parsing phase is completed, the resulting keys and values (or keys alone) are available for further processing. This processing is not carried out by the low-level parser itself, and so the parser requires the names of two functions along with the key–value list. One function is needed to process key–value pairs (it receives two arguments), and a second function is required for keys given without any value (it is called with a single argument).

The parser does not double # tokens or expand any input. Active tokens = and , appearing at the outer level of braces are converted to category “other” (12) so that the parser does not “miss” any due to category code changes. Spaces are removed from the ends of the keys and values. Keys and values which are given in braces will have exactly one set removed (after space trimming), thus

\[
\text{key} = \{\text{value here}\},
\]

and

\[
\text{key} = \text{value here},
\]

are treated identically.
Parses the \langle key–value list \rangle into a series of \langle keys \rangle and associated \langle values \rangle, or keys alone (if no \langle value \rangle was given). \langle function_1 \rangle should take one argument, while \langle function_2 \rangle should absorb two arguments. After \keyval_parse:NNn has parsed the \langle key–value list \rangle, \langle function_1 \rangle will be used to process keys given with no value and \langle function_2 \rangle will be used to process keys given with a value. The order of the \langle keys \rangle in the \langle key–value list \rangle will be preserved. Thus

\keyval_parse:NNn \function:n \function:nn
\{ key1 = value1 , key2 = value2, key3 = , key4 \}

will be converted into an input stream

\function:nn { key1 } { value1 }
\function:nn { key2 } { value2 }
\function:nn { key3 } { }
\function:n { key4 }

Note that there is a difference between an empty value (an equals sign followed by nothing) and a missing value (no equals sign at all). Spaces are trimmed from the ends of the \langle key \rangle and \langle value \rangle, then one outer set of braces is removed from the \langle key \rangle and \langle value \rangle as part of the processing.
Part XXI
The l3file package
File and I/O operations

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{} will attempt to locate them 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 \langle file name \rangle argument, this argument may contain both literal items and expandable content, which should on full expansion be the desired file name. Any active characters (as declared in \l_char_active_seq) will not be expanded, allowing the direct use of these in file names. File names will be quoted using " tokens if they contain spaces: as a result, " tokens are not permitted in file names.

1 File operation functions

\g_file_current_name_tl
Contains the name of the current \LaTeX{} file. This variable should not be modified: it is intended for information only. It will be equal to \c_job_name_tl at the start of a \LaTeX{} run and will be modified each time a file is read using \file_input:n.

\file_if_exist:nTF
\file_if_exist:nTF \langle file name \rangle \{\langle true code \rangle\} \{\langle false code \rangle\}
Updated: 2012-02-10
Searches for \langle file name \rangle using the current \TeX{} search path and the additional paths controlled by \file_path_include:n).

\file_add_path:nN
\file_add_path:nN \langle file name \rangle \{tl var\}
Updated: 2012-02-10
Searches for \langle file name \rangle in the path as detailed for \file_if_exist:nTF, and if found sets the \langle tl var \rangle the fully-qualified name of the file, i.e. the path and file name. If the file is not found then the \langle tl var \rangle will contain the marker \q_no_value.

\file_input:n
\file_input:n \langle file name \rangle
Updated: 2012-02-17
Searches for \langle file name \rangle in the path as detailed for \file_if_exist:nTF, and if found reads in the file as additional \LaTeX{} source. All files read are recorded for information and the file name stack is updated by this function. An error will be raised if the file is not found.
\file_list:

This function will list all files loaded using \file_input:n in the log file.

1.1 Input–output stream management

As \TeX{} is limited to 16 input streams and 16 output streams, direct use of the streams
by the programmer is not supported in \LaTeX{}3. 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.

\ior_open:NnTF ⟨stream⟩ {⟨file name⟩} {⟨true code⟩} {⟨false code⟩}

Opens \langle file name ⟩ for reading using \langle stream ⟩ as the control sequence for file access. If the
\langle stream ⟩ was already open it is closed before the new operation begins. The \langle stream ⟩
is available for access immediately and will remain allocated to \langle file name ⟩ until a \ior_close:N
instruction is given or the \TeX{} run ends. The \langle true code ⟩ is then inserted into the input stream. If the file is not found, no error is raised and the \langle false code ⟩ is inserted into the input stream.
\iow_open:Nn
\iow_open:cn

Opens \textit{file name} for writing using \textit{stream} as the control sequence for file access. If the \textit{stream} was already open it is closed before the new operation begins. The \textit{stream} is available for access immediately and will remain allocated to \textit{file name} until a \iow_close:N instruction is given or the \TeX{} run ends. Opening a file for writing will clear any existing content in the file (\textit{i.e.} writing is not additive).

\ior_close:N
\ior_close:N
\ior_close:cn
\ior_close:cn

Closes the \textit{stream}. Streams should always be closed when they are finished with as this ensures that they remain available to other programmers.

\ior_list_streams:
\ior_list_streams:
\ior_list_streams:
\ior_list_streams:

Displays a list of the file names associated with each open stream: intended for tracking down problems.

1.2 Reading from files

\ior_get:NN
\ior_get:NN

Function that reads one or more lines (until an equal number of left and right braces are found) from the input \textit{stream} and stores the result locally in the \textit{token list} variable. If the \textit{stream} is not open, input is requested from the terminal. The material read from the \textit{stream} will be tokenized by \TeX{} according to the category codes in force when the function is used. Note that any blank lines will be converted to the token \par. Therefore, if skipping blank lines is requires a test such as

\begin{verbatim}
\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 \l_tmpb_tl
... 
\end{verbatim}

may be used. Also notice that if multiple lines are read to match braces then the resulting token list will contain \par tokens. As normal \TeX{} tokenization is in force, any lines which do not end in a comment character (usually \%) will have the line ending converted to a space, so for example input

\begin{verbatim}
a b c
\end{verbatim}

will result in a token list \texttt{a b c}.

\textbf{\TeX{}hackers note:} This protected macro expands to the \TeX{} primitive \texttt{\read} along with the \texttt{to} keyword.
\ior_get_str:NN \ior_get_str:NN ⟨stream⟩ ⟨token list variable⟩

Function that reads one line from the input ⟨stream⟩ and stores the result locally in the ⟨token list⟩ variable. If the ⟨stream⟩ is not open, input is requested from the terminal. The material is read from the ⟨stream⟩ 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 will always only read one line and any blank lines in the input will result in the ⟨token list variable⟩ being empty. Unlike \ior_get:N, line ends do not receive any special treatment. Thus input

\begin{verbatim}
a b c
\end{verbatim}

will result in a token list \begin{verbatim}a b c\end{verbatim} with the letters \begin{verbatim}a\end{verbatim}, \begin{verbatim}b\end{verbatim}, and \begin{verbatim}c\end{verbatim} having category code 12.

**\TeXhackers note:** This protected macro is a wrapper around the $\varepsilon$-$\TeX$ primitive \texttt{\readline}. However, the end-line character normally added by this primitive is not included in the result of \ior_get_str:NN.

\ior_if_eof_p:N \ior_if_eof_p:N ⟨stream⟩
\ior_if_eof:NTF \ior_if_eof:NTF ⟨stream⟩ {(true code)} {(false code)}

Tests if the end of a ⟨stream⟩ has been reached during a reading operation. The test will also return a true value if the ⟨stream⟩ is not open.

2 Writing to files

\ior_now:Nn \ior_now:Nn ⟨stream⟩ ⟨tokens⟩

This functions writes ⟨tokens⟩ to the specified ⟨stream⟩ immediately (i.e. the write operation is called on expansion of \ior_now:Nn).

\ior_log:n \ior_log:n ⟨tokens⟩

This function writes the given ⟨tokens⟩ to the log (transcript) file immediately: it is a dedicated version of \ior_now:Nn.

\ior_term:n \ior_term:n ⟨tokens⟩

This function writes the given ⟨tokens⟩ to the terminal file immediately: it is a dedicated version of \ior_now:Nn.
\iow_shipout:Nn \iow_shipout:Nn ⟨stream⟩ ⟨{tokens}⟩

This function writes ⟨tokens⟩ to the specified ⟨stream⟩ when the current page is finalised (i.e. at shipout). The x-type variants expand the ⟨tokens⟩ at the point where the function is used but not when the resulting tokens are written to the ⟨stream⟩ (cf. \iow_shipout_x:Nn).

\TeXhacksnote: When using expl3 with a format other than \LaTeX, new line characters inserted using \iow_newline: or using the line-wrapping code \iow_wrap:nnnN will not be recognized in the argument of \iow_shipout:Nn. This may lead to the insertion of additional unwanted line-breaks.

\iow_shipout_x:Nn \iow_shipout_x:Nn ⟨stream⟩ ⟨{tokens}⟩

This function writes ⟨tokens⟩ to the specified ⟨stream⟩ when the current page is finalised (i.e. at shipout). The ⟨tokens⟩ 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).

\TeXhacksnote: This is a wrapper around the \TeX primitive \write. When using expl3 with a format other than \LaTeX, new line characters inserted using \iow_newline: or using the line-wrapping code \iow_wrap:nnnN will not be 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 ⟨char⟩ into the output stream. Useful when trying to write difficult characters such as %, {, }, etc. in messages, for example:

\iow_now:Nx \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 ⟨tokens⟩ 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).

\TeXhacksnote: When using expl3 with a format other than \LaTeX, the character inserted by \iow_newline: will not be recognized by \TeX, which may lead to the insertion of additional unwanted line-breaks. This issue only affects \iow_shipout:Nn, \iow_shipout_x:Nn and direct uses of primitive operations.
2.1 Wrapping lines in output

\iow_wrap:nnnN
\iow_wrap:nnnN {⟨text⟩} {⟨run-on text⟩} {⟨set up⟩} {⟨function⟩}

This function will wrap the ⟨text⟩ to a fixed number of characters per line. At the start of each line which is wrapped, the ⟨run-on text⟩ will be inserted. The line character count targeted will be the value of \l_iow_line_count_int minus the number of characters in the ⟨run-on text⟩. The ⟨text⟩ and ⟨run-on text⟩ are exhaustively expanded by the function, with the following substitutions:

- \ 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_indent:n may be used to indent a part of the message.

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

Any expandable material in the ⟨text⟩ 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 ⟨function⟩, which will typically be a wrapper around a write operation. The output of \iow_wrap:nnnN (i.e. the argument passed to the ⟨function⟩) will consist of characters of category “other” (category code 12), with the exception of spaces which will have category “space” (category code 10). This means that the output will not expand further when written to a file.

**\TeXhackers note:** Internally, \iow_wrap:nnnN carries out an x-type expansion on the ⟨text⟩ 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 ⟨text⟩.

\iow_indent:n
\iow_indent:n {⟨text⟩}

In the context of \iow_wrap:nnnN (for instance in messages), indents ⟨text⟩ by four spaces. This function will 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
\l_iow_line_count_int

The maximum number of characters in a line to be written by the \iow_wrap:nnnN function. This value depends on the \TeX system in use: the standard value is 78, which is typically correct for unmodified \TeXlive and \MiKTeX systems.
2.2 Constant input–output streams

\c_term_ior

Constant input stream for reading from the terminal. Reading from this stream using \ior_get:NN or similar will result in a prompt from \TeX of the form

\texttt{<tl>=}

\c_log_iow  \c_term_iow

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

2.3 Primitive conditionals

\if_eof:w * \if_eof:w \langle\text{stream}\rangle

\langle\text{true code}\rangle

\texttt{\else:}

\langle\text{false code}\rangle

\texttt{\fi:}

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

\TeXhackers note: This is the \TeX primitive \texttt{\ifeof}.

2.4 Internal file functions and variables

\g_file_internal_ior

Used to test for the existence of files when opening.

\l_file_internal_name_tl

Used to return the full name of a file for internal use. This is set by \texttt{\_file_if_exist:n(TF)} and \texttt{\_file_if_exist:nT}, and the value may then be used to load a file directly provided no further operations intervene.

\texttt{\_file_name_sanitize:nn \langle\text{name}\rangle \{\langle\text{tokens}\rangle\}}

Exhaustively expands the \langle\text{name}\rangle with the exception of any category \langle\text{active}\rangle (catcode 13) tokens, which are not expanded. The list of \langle\text{active}\rangle tokens is taken from \texttt{\l_char_active_seq}. The \langle\text{sanitized name}\rangle is then inserted (in braces) after the \langle\text{tokens}\rangle, which should further process the file name. If any spaces are found in the name after expansion, an error is raised.
2.5 Internal input–output functions

\_\_ior\_open:NN \_\_ior\_open:No

This function has identical syntax to the public version. However, it does not take precautions against active characters in the \langle file name \rangle, and it does not attempt to add a \langle path \rangle to the \langle file name \rangle: it is therefore intended to be used by higher-level functions which have already fully expanded the \langle file name \rangle and which need to perform multiple open or close operations. See for example the implementation of \texttt{file\_add\_path:NN},

\_\_iow\_with:NN

\_\_iow\_with:No

New: 2012–01–23

If the \langle integer \rangle is equal to the \langle value \rangle then this function simply runs the \langle code \rangle. Otherwise it saves the current value of the \langle integer \rangle, sets it to the \langle value \rangle, runs the \langle code \rangle, and restores the \langle integer \rangle to its former value. This is used to ensure that the \texttt{\newlinechar} is 10 when writing to a stream, which lets \texttt{\iow\_newline} work, and that \texttt{\errorcontextlines} is $-1$ when displaying a message.
Part XXII

The l3fp package: floating points

A decimal floating point number is one which is stored as a significand and a separate exponent. The module implements expandably a wide set of arithmetic, trigonometric, and other operations on decimal floating point numbers, to be used within floating point expressions. Floating point expressions support the following operations with their usual precedence.

- Basic arithmetic: addition $x + y$, subtraction $x - y$, multiplication $x \times y$, division $x/y$, square root $\sqrt{x}$, and parentheses.
- Comparison operators: $x < y$, $x \leq y$, $x > y$, $x \geq y$, $x = y$ etc.
- Boolean logic: negation $!x$, conjunction $x \& \& y$, disjunction $x \mid \mid y$, ternary operator $x ? y : z$.
- Exponentials: $\exp x$, $\ln x$, $x^y$.
- Trigonometry: $\sin x$, $\cos x$, $\tan x$, $\cot x$, $\sec x$, $\csc x$ expecting their arguments in radians, and $\sin d x$, $\cos d x$, $\tan d x$, $\cot d x$, $\sec d x$, $\csc d x$ expecting their arguments in degrees.
- Inverse trigonometric functions: $\arcsin x$, $\arccos x$, $\arctan x$, $\arccot x$, $\arcsec x$, $\arccsc x$ giving a result in radians, and $\arcsind x$, $\arccosd x$, $\arctand x$, $\arccotd x$, $\arcsecd x$, $\arccscd x$ giving a result in degrees.
- Extrema: $\max(x, y, ...)$, $\min(x, y, ...)$, $\abs(x)$.
- Rounding functions: $\text{round}(x, n)$ rounds to closest, $\text{trunc}(x, n)$ rounds towards zero, $\text{floor}(x, n)$ rounds towards $-\infty$, $\text{ceil}(x, n)$ rounds towards $+\infty$. And (not yet) modulo, and “quantize”.
- Constants: $\pi$, $\deg$ (one degree in radians).
- Dimensions, automatically expressed in points, e.g., $\text{pc}$ is 12.
- Automatic conversion (no need for \langle\text{type}\rangle_{\text{use}:N}) of integer, dimension, and skip variables to floating points, expressing dimensions in points and ignoring the stretch and shrink components of skips.

Floating point numbers can be given either explicitly (in a form such as $1.234e-34$, or $-0.0001$), or as a stored floating point variable, which is automatically replaced by its current value. See section 9.1 for a description of what a floating point is, section 9.2 for details about how an expression is parsed, and section 9.3 to know what the various operations do. Some operations may raise exceptions (error messages), described in section 7.

An example of use could be the following.
\texttt{\LaTeX{}} can now compute: $ \frac{\sin (3.5)}{2} + 2 \cdot 10^{-3} = \texttt{\ExplSyntaxOn \fp_to_decimal:n \{\sin 3.5 / 2 + 2e-3\} \texttt{\ExplSyntaxOff}}$.

But in all fairness, this module is mostly meant as an underlying tool for higher-level commands. For example, one could provide a function to typeset nicely the result of floating point computations.

\usepackage{xparse, siunitx}
\ExplSyntaxOn
\NewDocumentCommand { \calcnum } { m } { \num { \fp_to_scientific:n {#1} } }
\ExplSyntaxOff
\calcnum { 2 \pi \ast \sin ( 2.3 ^ 5 ) }

1 Creating and initialising floating point variables

\texttt{\fp_new:N} \langle \texttt{fp var} \rangle

Creates a new \langle \texttt{fp var} \rangle or raises an error if the name is already taken. The declaration is global. The \langle \texttt{fp var} \rangle will initially be +0.

\texttt{\fp_const:Nn} \langle \texttt{fp var} \rangle \{\langle \texttt{floating point expression} \rangle\}

Creates a new constant \langle \texttt{fp var} \rangle or raises an error if the name is already taken. The \langle \texttt{fp var} \rangle will be set globally equal to the result of evaluating the \langle \texttt{floating point expression} \rangle.

\texttt{\fp_zero:N} \langle \texttt{fp var} \rangle

Sets the \langle \texttt{fp var} \rangle to +0.

\texttt{\fp_zero_new:N} \langle \texttt{fp var} \rangle

Ensures that the \langle \texttt{fp var} \rangle exists globally by applying \texttt{\fp_new:N} if necessary, then applies \texttt{\fp_(g)zero:N} to leave the \langle \texttt{fp var} \rangle set to +0.

2 Setting floating point variables

\texttt{\fp_set:Nn} \langle \texttt{fp var} \rangle \{\langle \texttt{floating point expression} \rangle\}

Sets \langle \texttt{fp var} \rangle equal to the result of computing the \langle \texttt{floating point expression} \rangle.
\texttt{\textbackslash fp\_set\_eq:NN} \hspace{1cm} \texttt{\textbackslash fp\_set\_eq:NN}\langle \text{fp\ var}\rangle \langle \text{fp\ var}_2\rangle

Sets the floating point variable \langle \text{fp\ var}\rangle equal to the current value of \langle \text{fp\ var}_2\rangle.

\texttt{\textbackslash fp\_add:NN} \hspace{1cm} \texttt{\textbackslash fp\_add:NN}\langle \text{fp\ var}\rangle \{\langle\text{floating\ point\ expression}\rangle\}

Adds the result of computing the \langle\text{floating\ point\ expression}\rangle to the \langle \text{fp\ var}\rangle.

\texttt{\textbackslash fp\_sub:NN} \hspace{1cm} \texttt{\textbackslash fp\_sub:NN}\langle \text{fp\ var}\rangle \{\langle\text{floating\ point\ expression}\rangle\}

Subtracts the result of computing the \langle\text{floating\ point\ expression}\rangle from the \langle \text{fp\ var}\rangle.

\section{Using floating point numbers}

\texttt{\textbackslash fp\_eval:n} \hspace{1cm} \texttt{\textbackslash fp\_eval:n}\{\langle\text{floating\ point\ expression}\rangle\}

Evaluates the \langle\text{floating\ point\ expression}\rangle and expresses the result as a decimal number with no exponent. Leading or trailing zeros may be inserted to compensate for the exponent. Non-significant trailing zeros are trimmed, and integers are expressed without a decimal separator. The values $\pm\infty$ and NaN trigger an “invalid operation” exception. This function is identical to \texttt{\textbackslash fp\_to\_decimal:n}.

\texttt{\textbackslash fp\_to\_decimal:N} \hspace{1cm} \texttt{\textbackslash fp\_to\_decimal:N}\langle \text{fp\ var}\rangle \hspace{1cm} \texttt{\textbackslash fp\_to\_decimal:n}\{\langle\text{floating\ point\ expression}\rangle\}

Evaluates the \langle\text{floating\ point\ expression}\rangle and expresses the result as a decimal number with no exponent. Leading or trailing zeros may be inserted to compensate for the exponent. Non-significant trailing zeros are trimmed, and integers are expressed without a decimal separator. The values $\pm\infty$ and NaN trigger an “invalid operation” exception.

\texttt{\textbackslash fp\_to\_dim:N} \hspace{1cm} \texttt{\textbackslash fp\_to\_dim:N}\langle \text{fp\ var}\rangle \hspace{1cm} \texttt{\textbackslash fp\_to\_dim:n}\{\langle\text{floating\ point\ expression}\rangle\}

Evaluates the \langle\text{floating\ point\ expression}\rangle and expresses the result as a dimension (in pt) suitable for use in dimension expressions. The output is identical to \texttt{\textbackslash fp\_to\_decimal:n}, with an additional trailing pt. In particular, the result may be outside the range $[-2^{14}+2^{-17},2^{14}-2^{-17}]$ of valid \TeX\ dimensions, leading to overflow errors if used as a dimension. The values $\pm\infty$ and NaN trigger an “invalid operation” exception.
\texttt{\fp_to_int:N} \texttt{\fp_var} \texttt{\langle floating point expression\rangle}
\texttt{\fp_to_int:n \{\langle floating point expression\rangle\}}

Evaluates the \texttt{\langle floating point expression\rangle}, and rounds the result to the closest integer, rounding exact ties to an even integer. The result may be outside the range \([-2^{31} + 1, 2^{31} - 1]\) of valid \TeX integers, leading to overflow errors if used in an integer expression. The values $\pm\infty$ and \texttt{NaN} trigger an “invalid operation” exception.

\texttt{\fp_to_scientific:N} \texttt{\fp_var} \texttt{\langle floating point expression\rangle}
\texttt{\fp_to_scientific:n \{\langle floating point expression\rangle\}}

Evaluates the \texttt{\langle floating point expression\rangle} and expresses the result in scientific notation: \langle optional -\rangle\langle digit\rangle\langle 15 digits\rangle e\langle optional sign\rangle\langle exponent\rangle

The leading \langle digit\rangle is non-zero except in the case of $\pm 0$. The values $\pm\infty$ and \texttt{NaN} trigger an “invalid operation” exception.

\texttt{\fp_to_tl:N} \texttt{\fp_var} \texttt{\langle floating point expression\rangle}
\texttt{\fp_to_tl:n \{\langle floating point expression\rangle\}}

Evaluates the \texttt{\langle floating point expression\rangle} and expresses the result in (almost) the shortest possible form. Numbers in the ranges $(0, 10^{-3})$ and $[10^{16}, \infty)$ are expressed in scientific notation with trailing zeros trimmed and no decimal separator when there is a single significant digit (see \texttt{\fp_to_scientific:n}). Numbers in the range $[10^{-3}, 10^{16})$ are expressed in a decimal notation without exponent, with trailing zeros trimmed, and no decimal separator for integer values (see \texttt{\fp_to_decimal:n}). Negative numbers start with -. The special values $\pm 0$, $\pm\infty$ and \texttt{NaN} are rendered as 0, $-0$, \texttt{inf}, $-\texttt{inf}$, and \texttt{nan} respectively.

\texttt{\fp_use:N} \texttt{\fp_var}
\texttt{\fp_use:n \{\langle floating point expression\rangle\}}

Inserts the value of the \texttt{\langle fp var\rangle} into the input stream as a decimal number with no exponent. Leading or trailing zeros may be inserted to compensate for the exponent. Non-significant trailing zeros are trimmed. Integers are expressed without a decimal separator. The values $\pm 0$, $\pm\infty$ and \texttt{NaN} trigger an “invalid operation” exception. This function is identical to \texttt{\fp_to_decimal:N}.

4 Floating point conditionals

\texttt{\fp_if_exist_p:N} \texttt{\fp_var}
\texttt{\fp_if_exist:NTF \fp_var \{\langle true code\rangle\} \{\langle false code\rangle\}}

Tests whether the \texttt{\langle fp var\rangle} is currently defined. This does not check that the \texttt{\langle fp var\rangle} really is a floating point variable.
\fp_compare:p:nNn  \fp_compare:p:nNn \{\fpexpr\} \{\rel\} \{\fpexpr\}
\fp_compare:p:nNnTF \{\fpexpr\} \{\rel\} \{\fpexpr\} \{\true\} \{\false\}

Compares the \( \fpexpr_1 \) and the \( \fpexpr_2 \), and returns \true if the \( \rel \) is obeyed. Two floating point numbers \( x \) and \( y \) may obey four mutually exclusive relations: \( x \langle y, x=y, x \rangle y \), or \( x \) and \( y \) are not ordered. The latter case occurs exactly when either operand is NaN, and this relation is denoted by the symbol \?. Note that a NaN is distinct from any value, even another NaN, hence \( x=x \) is not true for a NaN. To test if a value is NaN, compare it to an arbitrary number with the “not ordered” relation.

\fp_compare:nNnTF \{<value>\} \{0\}
\{\} \% <value> is nan
\{\} \% <value> is not nan

\fp_compare:p:n
\fp_compare:p:TF
\fp_compare:p
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_compare:n
\fp_com
5 Floating point expression loops

\fp_do_until:nNnn \$\fp_do_until:nNnn \{\text{\textit{fpexpr}}\_1\} \{\text{\textit{relation}}\} \{\text{\textit{fpexpr}}\_2\} \{\langle \text{\textit{code}}\rangle\}$

Places the \langle \text{\textit{code}}\rangle in the input stream for \TeX to process, and then evaluates the relationship between the two \langle \text{\textit{floating point expressions}}\rangle as described for \fp_compare:nNnTF. If the test is \textit{false} then the \langle \text{\textit{code}}\rangle will be inserted into the input stream again and a loop will occur until the \langle \text{\textit{relation}}\rangle is \textit{true}.

\fp_do_while:nNnn \$\fp_do_while:nNnn \{\text{\textit{fpexpr}}\_1\} \{\text{\textit{relation}}\} \{\text{\textit{fpexpr}}\_2\} \{\langle \text{\textit{code}}\rangle\}$

Places the \langle \text{\textit{code}}\rangle in the input stream for \TeX to process, and then evaluates the relationship between the two \langle \text{\textit{floating point expressions}}\rangle as described for \fp_compare:nNnTF. If the test is \textit{true} then the \langle \text{\textit{code}}\rangle will be inserted into the input stream again and a loop will occur until the \langle \text{\textit{relation}}\rangle is \textit{false}.

\fp_until_do:nNnn \$\fp_until_do:nNnn \{\text{\textit{fpexpr}}\_1\} \{\text{\textit{relation}}\} \{\text{\textit{fpexpr}}\_2\} \{\langle \text{\textit{code}}\rangle\}$

Evaluates the relationship between the two \langle \text{\textit{floating point expressions}}\rangle as described for \fp_compare:nNnTF, and then places the \langle \text{\textit{code}}\rangle in the input stream if the \langle \text{\textit{relation}}\rangle is \textit{false}. After the \langle \text{\textit{code}}\rangle has been processed by \TeX the test will be repeated, and a loop will occur until the test is \textit{true}.

\fp_while_do:nNnn \$\fp_while_do:nNnn \{\text{\textit{fpexpr}}\_1\} \{\text{\textit{relation}}\} \{\text{\textit{fpexpr}}\_2\} \{\langle \text{\textit{code}}\rangle\}$

Evaluates the relationship between the two \langle \text{\textit{floating point expressions}}\rangle as described for \fp_compare:nNnTF, and then places the \langle \text{\textit{code}}\rangle in the input stream if the \langle \text{\textit{relation}}\rangle is \textit{true}. After the \langle \text{\textit{code}}\rangle has been processed by \TeX the test will be repeated, and a loop will occur until the test is \textit{false}.

\fp_do_until:nn \$\fp_do_until:nn \{\text{\textit{fpexpr}}\_1\} \{\text{\textit{relation}}\} \{\text{\textit{fpexpr}}\_2\} \{\langle \text{\textit{code}}\rangle\}$

Places the \langle \text{\textit{code}}\rangle in the input stream for \TeX to process, and then evaluates the relationship between the two \langle \text{\textit{floating point expressions}}\rangle as described for \fp_compare:nTF. If the test is \textit{false} then the \langle \text{\textit{code}}\rangle will be inserted into the input stream again and a loop will occur until the \langle \text{\textit{relation}}\rangle is \textit{true}.

\fp_do_while:nn \$\fp_do_while:nn \{\text{\textit{fpexpr}}\_1\} \{\text{\textit{relation}}\} \{\text{\textit{fpexpr}}\_2\} \{\langle \text{\textit{code}}\rangle\}$

Places the \langle \text{\textit{code}}\rangle in the input stream for \TeX to process, and then evaluates the relationship between the two \langle \text{\textit{floating point expressions}}\rangle as described for \fp_compare:nTF. If the test is \textit{true} then the \langle \text{\textit{code}}\rangle will be inserted into the input stream again and a loop will occur until the \langle \text{\textit{relation}}\rangle is \textit{false}.

\fp_until_do:nn \$\fp_until_do:nn \{\text{\textit{fpexpr}}\_1\} \{\text{\textit{relation}}\} \{\text{\textit{fpexpr}}\_2\} \{\langle \text{\textit{code}}\rangle\}$

Evaluates the relationship between the two \langle \text{\textit{floating point expressions}}\rangle as described for \fp_compare:nTF, and then places the \langle \text{\textit{code}}\rangle in the input stream if the \langle \text{\textit{relation}}\rangle is \textit{false}. After the \langle \text{\textit{code}}\rangle has been processed by \TeX the test will be repeated, and a loop will occur until the test is \textit{true}.
\fp_while_do:nn \fp_while_do:nn \{ (fpexpr_1) \ (relation) \ (fpexpr_2) \} \{(code)\}

Evaluates the relationship between the two (floating point expressions) as described for \fp_compare:nTF, and then places the (code) in the input stream if the (relation) is true. After the (code) has been processed by \TeX{} the test will be repeated, and a loop will occur until the test is false.

6 Some useful constants, and scratch variables

\verbatim
\c_zero_fp
\c_minus_zero_fp
\verbatim
Zero, with either sign.

\verbatim
\c_one_fp
\verbatim
One as an fp: useful for comparisons in some places.

\verbatim
\c_inf_fp
\c_minus_inf_fp
\verbatim
Infinity, with either sign. These can be input directly in a floating point expression as inf and -inf.

\verbatim
\c_e_fp
\verbatim
The value of the base of the natural logarithm, $e = \exp(1)$.

\verbatim
\c_pi_fp
\verbatim
The value of $\pi$. This can be input directly in a floating point expression as pi.

\verbatim
\c_one_degree_fp
\verbatim
The value of $1^\circ$ in radians. Multiply an angle given in degrees by this value to obtain a result in radians. Note that trigonometric functions expecting an argument in radians or in degrees are both available. Within floating point expressions, this can be accessed as deg.

\verbatim
\l_tmpa_fp
\verbatim
Scratch floating points for local assignment. These are never used by the kernel code, and so are safe for use with any \TeX{}-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.

\verbatim
\l_tmpb_fp
\verbatim
Scratch floating points for global assignment. These are never used by the kernel code, and so are safe for use with any \TeX{}-defined function. However, they may be overwritten by other non-kernel code and so should only be used for short-term storage.
Chapter 7 Floating point exceptions

The functions defined in this section are experimental, and their functionality may be altered or removed altogether.

“Exceptions” may occur when performing some floating point operations, such as $0/0$, or $10^{10^{10}}$. The IEEE standard defines 5 types of exceptions.

- **Overflow** occurs whenever the result of an operation is too large to be represented as a normal floating point number. This results in $\pm\infty$.

- **Underflow** occurs whenever the result of an operation is too close to 0 to be represented as a normal floating point number. This results in $\pm0$.

- **Invalid operation** occurs for operations with no defined outcome, for instance $0/0$, or $\sin(\infty)$, and almost any operation involving a NaN. This normally results in a NaN, except for conversion functions whose target type does not have a notion of NaN (e.g., `\fp_to_dim:n`).

- **Division by zero** occurs when dividing a non-zero number by 0, or when evaluating e.g., $\ln(0)$ or $\cot(0)$. This results in $\pm\infty$.

- **Inexact** occurs whenever the result of a computation is not exact, in other words, almost always. At the moment, this exception is entirely ignored in LaTeX3.

To each exception is associated a “flag”, which can be either on or off. By default, the “invalid operation” exception triggers an (expandable) error, and raises the corresponding flag. Other exceptions only raise the corresponding flag. The state of the flag can be tested and modified. The behaviour when an exception occurs can be modified (using `\fp_trap:nn`) to either produce an error and turn the flag on, or only turn the flag on, or do nothing at all.

```
\fp_if_flag_on_p:n {⟨exception⟩}
\fp_if_flag_on:nTF {⟨exception⟩} {⟨true code⟩} {⟨false code⟩}
```

Tests if the flag for the ⟨exception⟩ is on, which normally means the given ⟨exception⟩ has occurred. This function is experimental, and may be altered or removed.

```
\fp_flag_off:n {⟨exception⟩}
```

Locally turns off the flag which indicates whether the ⟨exception⟩ has occurred. This function is experimental, and may be altered or removed.

```
\fp_flag_on:n {⟨exception⟩}
```

Locally turns on the flag to indicate (or pretend) that the ⟨exception⟩ has occurred. Note that this function is expandable: it is used internally by 13fp to signal when exceptions do occur. This function is experimental, and may be altered or removed.
All occurrences of the \langle exception\rangle (invalid_operation, division_by_zero, overflow, or underflow) within the current group are treated as \langle trap type\rangle, which can be

- none: the \langle exception\rangle will be entirely ignored, and leave no trace;
- flag: the \langle exception\rangle will turn the corresponding flag on when it occurs;
- error: additionally, the \langle exception\rangle will halt the \TeX run and display some information about the current operation in the terminal.

This function is experimental, and may be altered or removed.

8 Viewing floating points

9 Floating point expressions

9.1 Input of floating point numbers

We support four types of floating point numbers:

- \( \pm 0.d_1d_2\ldots d_{16} \cdot 10^n \), a normal floating point number, with \( d_i \in [0, 9] \), \( d_1 \neq 0 \), and \( |n| \leq 10000 \);
- \( \pm 0 \), zero, with a given sign;
- \( \pm \infty \), infinity, with a given sign;
- NaN, is “not a number”, and can be either quiet or signalling (not yet: this distinction is currently unsupported);

(not yet) subnormal numbers \( \pm 0.d_1d_2\ldots d_{16} \cdot 10^{-10000} \) with \( d_1 = 0 \).

Normal floating point numbers are stored in base 10, with 16 significant figures.

On input, a normal floating point number consists of:

- \langle sign\rangle: a possibly empty string of + and - characters;
- \langle significand\rangle: a non-empty string of digits together with zero or one dot;
- \langle exponent\rangle optionally: the character e, followed by a possibly empty string of + and - tokens, and a non-empty string of digits.
The sign of the resulting number is + if \( \langle \text{sign} \rangle \) contains an even number of \(-\), and \(-\) otherwise, hence, an empty \( \langle \text{sign} \rangle \) denotes a non-negative input. The stored significand is obtained from \( \langle \text{significand} \rangle \) by omitting the decimal separator and leading zeros, and rounding to 16 significant digits, filling with trailing zeros if necessary. In particular, the value stored is exact if the input \( \langle \text{significand} \rangle \) has at most 16 digits. The stored \( \langle \text{exponent} \rangle \) is obtained by combining the input \( \langle \text{exponent} \rangle \) (0 if absent) with a shift depending on the position of the significand and the number of leading zeros.

A special case arises if the resulting \( \langle \text{exponent} \rangle \) is either too large or too small for the floating point number to be represented. This results either in an overflow (the number is then replaced by \( \pm \infty \)), or an underflow (resulting in \( \pm 0 \)). The result is thus \( \pm 0 \) if and only if \( \langle \text{significand} \rangle \) contains no non-zero digit (\( i.e. \), consists only in 0 characters, and an optional . character), or if there is an underflow. Note that a single dot is currently a valid floating point number, equal to +0, but that is not guaranteed to remain true.

Special numbers are input as follows:

- **inf** represents \( +\infty \), and can be preceded by any \( \langle \text{sign} \rangle \), yielding \( \pm \infty \) as appropriate.
- **nan** represents a (quiet) non-number. It can be preceded by any sign, but that will be ignored.
- Any unrecognizable string triggers an error, and produces a NaN.

Note that \( e-1 \) is not a representation of \( 10^{-1} \), because it could be mistaken with the difference of “e” and 1. This is consistent with several other programming languages. However, in order to avoid confusions, \( e-1 \) is not considered to be this difference either. To input the base of natural logarithms, use \( \exp(1) \) or \c_e_fp.

### 9.2 Precedence of operators

We list here all the operations supported in floating point expressions, in order of decreasing precedence: operations listed earlier bind more tightly than operations listed below them.

- Function calls \( (\sin, \ln, \text{etc}) \).
- Binary ** and ^ (right associative).
- Unary +, -, !.
- Binary *, /, and implicit multiplication by juxtaposition \( (2\pi, 3(4+5), \text{etc}) \).
- Binary + and -. 
- Comparisons \( >, \geq, \neq, <, \text{ etc} \).
- Logical and, denoted by \&\&.
- Logical or, denoted by \mid\mid.
- Ternary operator \? (right associative).
The precedence of operations can be overridden using parentheses. In particular, those precedences imply that

\[
\sin 2\pi = \sin(2\pi) = 0,
\]

\[
2^{\max(3,4)} = 2^{2\max(3,4)} = 256.
\]

Functions are called on the value of their argument, contrarily to \TeX\ macros.

## 9.3 Operations

We now present the various operations allowed in floating point expressions, from the lowest precedence to the highest. When used as a truth value, a floating point expression is `false` if it is ±0, and `true` otherwise, including when it is NaN.

\[
\texttt{\textbackslash fp\_eval:n \{ \langle operand_1 \rangle \ ? \langle operand_2 \rangle : \langle operand_3 \rangle \ }}
\]

The ternary operator `?:` results in `\langle operand_2 \rangle` if `\langle operand_1 \rangle` is true, and `\langle operand_3 \rangle` if it is false (equal to ±0). All three `\langle operands \rangle` are evaluated in all cases. The operator is right associative, hence

\[
\texttt{\textbackslash fp\_eval:n}
\]

\[
\{
1 + 3 > 4 \ ? \ 1 : \\
2 + 4 > 5 \ ? \ 2 : \\
3 + 5 > 6 \ ? \ 3 : 4
\}
\]

first tests whether \(1 + 3 > 4\); since this isn’t true, the branch following `:` is taken, and \(2 + 4 > 5\) is compared; since this is true, the branch before `:` is taken, and everything else is (evaluated then) ignored. That allows testing for various cases in a concise manner, with the drawback that all computations are made in all cases.

\[
\texttt{\textbackslash fp\_eval:n \{ \langle operand_1 \rangle \  \&\& \ \langle operand_2 \rangle \ }}
\]

If `\langle operand_1 \rangle` is true (non-zero), use that value, otherwise the value of `\langle operand_2 \rangle`. Both `\langle operands \rangle` are evaluated in all cases.

\[
\texttt{\textbackslash fp\_eval:n \{ \langle operand_1 \rangle \  \&\& \ \langle operand_2 \rangle \ }}
\]

If `\langle operand_1 \rangle` is false (equal to ±0), use that value, otherwise the value of `\langle operand_2 \rangle`. Both `\langle operands \rangle` are evaluated in all cases.
Each ⟨relation⟩ consists of a non-empty string of <, =, >, and ?, optionally preceded by !, and may not start with ?. This evaluates to +1 if all comparisons ⟨operand⟩₁ ⟨relation⟩₁ ⟨operand⟩₂ are true, and +0 otherwise. All ⟨operands⟩ are evaluated in all cases. See \fp_compare:nTF for details.

+$ \fp_eval:n { \langle \text{operand}_1 \rangle + \langle \text{operand}_2 \rangle }$

-$ \fp_eval:n { \langle \text{operand}_1 \rangle - \langle \text{operand}_2 \rangle }$

Computes the sum or the difference of its two ⟨operands⟩. The “invalid operation” exception occurs for $\infty - \infty$. “Underflow” and “overflow” occur when appropriate.

* $ \fp_eval:n { \langle \text{operand}_1 \rangle * \langle \text{operand}_2 \rangle }$

/ $ \fp_eval:n { \langle \text{operand}_1 \rangle / \langle \text{operand}_2 \rangle }$

Computes the product or the ratio of its two ⟨operands⟩. The “invalid operation” exception occurs for $\infty/\infty$, $0/0$, or $0 * \infty$. “Division by zero” occurs when dividing a finite non-zero number by ±0. “Underflow” and “overflow” occur when appropriate.

+$ \fp_eval:n { + \langle \text{operand} \rangle }$

-$ \fp_eval:n { - \langle \text{operand} \rangle }$

! $ \fp_eval:n { ! \langle \text{operand} \rangle }$

The unary + does nothing, the unary - changes the sign of the ⟨operand⟩, and ! ⟨operand⟩ evaluates to 1 if ⟨operand⟩ is false and 0 otherwise (this is the not boolean function). Those operations never raise exceptions.

** $ \fp_eval:n { \langle \text{operand}_1 \rangle ^\langle \text{operand}_2 \rangle }$

$ \fp_eval:n { \langle \text{operand}_1 \rangle ^\langle \text{operand}_2 \rangle }$

Graph raises ⟨operand⟩₁ to the power ⟨operand⟩₂. This operation is right associative, hence 2 ** 2 ** 3 equals $2^2 = 256$. The “invalid operation” exception occurs if ⟨operand⟩₁ is negative or –0, and ⟨operand⟩₂ is not an integer, unless the result is zero (in that case, the sign is chosen arbitrarily to be +0). “Division by zero” occurs when raising ±0 to a strictly negative power. “Underflow” and “overflow” occur when appropriate.

abs $ \fp_eval:n { \text{abs} ( \langle \text{fexpr} \rangle ) }$

Computes the absolute value of the ⟨fexpr⟩. This function does not raise any exception beyond those raised when computing its operand ⟨fexpr⟩. See also \fp_abs:n.

exp $ \fp_eval:n { \text{exp} ( \langle \text{fexpr} \rangle ) }$

Computes the exponential of the ⟨fexpr⟩. “Underflow” and “overflow” occur when appropriate.
\fp_eval:n \{ ln( \langle fexpr \rangle ) \}

Computes the natural logarithm of the \langle fexpr \rangle. Negative numbers have no (real) logarithm, hence the “invalid operation” is raised in that case, including for ln(-0). “Division by zero” occurs when evaluating ln(+0) = -\infty. “Underflow” and “overflow” occur when appropriate.

\fp_eval:n \{ max( \langle fexpr \rangle_1, \langle fexpr \rangle_2, \ldots ) \}
\fp_eval:n \{ min( \langle fexpr \rangle_1, \langle fexpr \rangle_2, \ldots ) \}

Evaluates each \langle fexpr \rangle and computes the largest (smallest) of those. If any of the \langle fexpr \rangle is a NaN, the result is NaN. Those operations do not raise exceptions.

\fp_eval:n \{ round ( \langle fexpr \rangle ) \}
\fp_eval:n \{ round ( \langle fexpr \rangle_1, \langle fexpr \rangle_2 ) \}

Evaluates \langle fexpr \rangle_1 = x and \langle fexpr \rangle_2 = n, then rounds x to n places. If n is an integer, this rounds x to a multiple of 10^{-n}; if n = +\infty, this always yields x; if n = -\infty, this yields one of ±0, ±\infty, or NaN; if n is neither ±\infty nor an integer, then an “invalid operation” exception is raised. When \langle fexpr \rangle_2 is omitted, n = 0, i.e., \langle fexpr \rangle_1 is rounded to an integer. The rounding direction depends on the function:

- **round** yields the multiple of 10^{-n} closest to x, and if x is half-way between two such multiples, the even multiple is chosen (“ties to even”);
- **floor**, or the deprecated **round-**, yields the largest multiple of 10^{-n} smaller or equal to x (“round towards negative infinity”);
- **ceil**, or the deprecated **round+**, yields the smallest multiple of 10^{-n} greater or equal to x (“round towards positive infinity”);
- **trunc**, or the deprecated **round0**, yields a multiple of 10^{-n} with the same sign as x and with the largest absolute value less that that of x (“round towards zero”).

“Overflow” occurs if x is finite and the result is infinite (this can only happen if \langle fexpr \rangle_2 < -9984).

\fp_eval:n \{ sin( \langle fexpr \rangle ) \}
\fp_eval:n \{ cos( \langle fexpr \rangle ) \}
\fp_eval:n \{ tan( \langle fexpr \rangle ) \}
\fp_eval:n \{ cot( \langle fexpr \rangle ) \}
\fp_eval:n \{ csc( \langle fexpr \rangle ) \}
\fp_eval:n \{ sec( \langle fexpr \rangle ) \}

Computes the sine, cosine, tangent, cotangent, cosecant, or secant of the \langle fexpr \rangle given in radians. For arguments given in degrees, see sind, cosd, etc. Note that since \pi is irrational, sin(8\pi) is not quite zero, while its analog sind(8 \times 180) is exactly zero. The trigonometric functions are undefined for an argument of ±\infty, leading to the “invalid operation” exception. Additionally, evaluating tangent, cotangent, cosecant, or secant at one of their poles leads to a “division by zero” exception. “Underflow” and “overflow” occur when appropriate.
Computes the sine, cosine, tangent, cotangent, cosecant, or secant of the \(\langle fpexpr\rangle\) given in degrees. For arguments given in radians, see \texttt{sin}, \texttt{cos}, etc. Note that since \(\pi\) is irrational, \(\sin(8\pi)\) is not quite zero, while its analog \(\text{sind}(8 \times 180)\) is exactly zero. The trigonometric functions are undefined for an argument of \(\pm \infty\), leading to the “invalid operation” exception. Additionally, evaluating tangent, cotangent, cosecant, or secant at one of their poles leads to a “division by zero” exception. “Underflow” and “overflow” occur when appropriate.

Computes the arcsine, arccosine, arccosecant, or arcsecant of the \(\langle fpexpr\rangle\) and returns the result in radians, in the range \([-\pi/2, \pi/2]\) for \texttt{asin} and \texttt{acsc} and \([0, \pi]\) for \texttt{acos} and \texttt{asec}. For a result in degrees, use \texttt{asind}, etc. If the argument of \texttt{asin} or \texttt{acos} lies outside the range \([-1, 1]\), or the argument of \texttt{acsc} or \texttt{asec} inside the range \((-1, 1)\), an “invalid operation” exception is raised. “Underflow” and “overflow” occur when appropriate.

Computes the arcsine, arccosine, arccosecant, or arcsecant of the \(\langle fpexpr\rangle\) and returns the result in degrees, in the range \([-90, 90]\) for \texttt{asin} and \texttt{acsc} and \([0, 180]\) for \texttt{acos} and \texttt{asec}. For a result in radians, use \texttt{asind}, etc. If the argument of \texttt{asin} or \texttt{acos} lies outside the range \([-1, 1]\), or the argument of \texttt{acsc} or \texttt{asec} inside the range \((-1, 1)\), an “invalid operation” exception is raised. “Underflow” and “overflow” occur when appropriate.
\begin{align*}
\text{atan} & \quad \text{\texttt{\{atan(\text{\texttt{\{fpexpr\}})}\}}} \\
\text{acot} & \quad \text{\texttt{\{atan(\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}})}\}}} \\
\text{New: 2013-11-02} & \quad \text{\texttt{\{acot(\text{\texttt{\{fpexpr\}})}\}}} \\
& \quad \text{\texttt{\{acot(\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}})}\}}} \\
\end{align*}

Those functions yield an angle in radians: \texttt{atan} and \texttt{acotd} are their analogs in degrees. The one-argument versions compute the arctangent or arccotangent of the \texttt{\{fpexpr\}}: arctangent takes values in the range \([-\pi/2, \pi/2]\), and arccotangent in the range \([0, \pi]\). The two-argument arctangent computes the angle in polar coordinates of the point with Cartesian coordinates \((\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}}))\): this is the arctangent of \((\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}}))\), possibly shifted by \(\pi\) depending on the signs of \((\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}}))\) and \((\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}}))\). The two-argument arccotangent computes the angle in polar coordinates of the point \((\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}}))\), equal to the arccotangent of \((\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}}))\), possibly shifted by \(\pi\). Both two-argument functions take values in the wider range \([-\pi, \pi]\). The ratio \((\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}})) / (\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}}))\) need not be defined for the two-argument arctangent: when both expressions yield \(\pm 0\), or when both yield \(\pm \infty\), the resulting angle is one of \(\{\pm \pi/4, \pm 3\pi/4\}\) depending on signs. Only the "underflow" exception can occur.

\begin{align*}
\text{atand} & \quad \text{\texttt{\{atand(\text{\texttt{\{fpexpr\}})}\}}} \\
\text{acotd} & \quad \text{\texttt{\{acotd(\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}})}\}}} \\
\text{New: 2013-11-02} & \quad \text{\texttt{\{acotd(\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}})}\}}} \\
& \quad \text{\texttt{\{acotd(\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}})}\}}} \\
\end{align*}

Those functions yield an angle in degrees: \texttt{atand} and \texttt{acotd} are their analogs in radians. The one-argument versions compute the arctangent or arccotangent of the \texttt{\{fpexpr\}}: arctangent takes values in the range \([-90, 90]\), and arccotangent in the range \([0, 180]\). The two-argument arctangent computes the angle in polar coordinates of the point with Cartesian coordinates \((\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}}))\): this is the arctangent of \((\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}}))\), possibly shifted by 180 depending on the signs of \((\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}}))\) and \((\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}}))\). The two-argument arccotangent computes the angle in polar coordinates of the point \((\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}}))\), equal to the arccotangent of \((\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}}))\), possibly shifted by 180. Both two-argument functions take values in the wider range \([-180, 180]\). The ratio \((\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}})) / (\text{\texttt{\{fpexpr\}}, \text{\texttt{\{fpexpr\}}}))\) need not be defined for the two-argument arctangent: when both expressions yield \(\pm 0\), or when both yield \(\pm \infty\), the resulting angle is one of \(\{\pm 45, \pm 135\}\) depending on signs. Only the "underflow" exception can occur.

\begin{align*}
\text{sqrt} & \quad \text{\texttt{\{sqrt(\text{\texttt{\{fpexpr\}})}\}}} \\
\text{New: 2013-12-14} & \quad \text{\texttt{\{sqrt(\text{\texttt{\{fpexpr\}})}\}}} \\
\end{align*}

Computes the square root of the \texttt{\{fpexpr\}}. The "invalid operation" is raised when the \texttt{\{fpexpr\}} is negative; no other exception can occur. Special values yield \(\sqrt{-0} = -0\), \(\sqrt{+0} = +0\), \(\sqrt{\pm \infty} = +\infty\) and \(\sqrt{\text{NaN}} = \text{NaN}\).

\begin{align*}
\text{inf} & \quad \text{The special values } +\infty, -\infty, \text{ and } \text{NaN} \text{ are represented as } \text{inf}, -\text{inf} \text{ and } \text{nan} \text{ (see } \text{c_-inf_fp, c_minus_inf_fp} \text{ and } \text{c_nan_fp}). \\
\text{nan} & \quad \text{The value of } \pi \text{ (see } \text{c_pi_fp}). \\
\text{pi} & \quad \text{The value of } 1^\circ \text{ in radians (see } \text{c_one_degree_fp}). \\
\text{deg} & \quad \text{The value of } 1^\circ \text{ in radians (see } \text{c_one_degree_fp}). \\
\end{align*}
Those units of measurement are equal to their values in pt, namely

\[
\begin{align*}
1 \text{ in} &= 72.27 \text{ pt} \\
1 \text{ pt} &= 1 \text{ pt} \\
1 \text{ pc} &= 12 \text{ pt} \\
1 \text{ cm} &= \frac{1}{2.54} \text{ in} = 28.45275590551181 \text{ pt} \\
1 \text{ mm} &= \frac{1}{25.4} \text{ in} = 2.845275590551181 \text{ pt} \\
1 \text{ dd} &= 0.376065 \text{ mm} = 1.0700856496063 \text{ pt} \\
1 \text{ cc} &= 12 \text{ dd} = 12.84010277952756 \text{ pt} \\
1 \text{ nd} &= 0.375 \text{ mm} = 1.066978346456693 \text{ pt} \\
1 \text{ nc} &= 12 \text{ nd} = 12.80374015748031 \text{ pt} \\
1 \text{ bp} &= \frac{1}{72} \text{ in} = 0.00375 \text{ pt} \\
1 \text{ sp} &= 2^{-16} \text{ pt} = 1.52587890625e-5 \text{ pt} .
\end{align*}
\]

The values of the (font-dependent) units \texttt{em} and \texttt{ex} are gathered from \TeX{} when the surrounding floating point expression is evaluated.

The following need to be done. I’ll try to time-order the items.

- Decide what exponent range to consider.
• Support signalling \texttt{nan}.

• Modulo and remainder, and rounding functions \texttt{quantize}, \texttt{quantize0}, \texttt{quantize+}, \texttt{quantize-}, \texttt{quantize=} \texttt{round=}. Should the modulo also be provided as (catcode 12) \%?

• \texttt{\textbackslash fp\_format:nn \{fpexpr\} \{format\}}, but what should \texttt{format} be? More general pretty printing?

• Add \texttt{and}, \texttt{or}, \texttt{xor}? Perhaps under the names \texttt{all}, \texttt{any}, and \texttt{xor}?

• Add \texttt{log(x,b)} for logarithm of \texttt{x} in base \texttt{b}.

• \texttt{hypot} (Euclidean length). Cartesian-to-polar transform.

• Hyperbolic functions \texttt{cosh}, \texttt{sinh}, \texttt{tanh}.

• Inverse hyperbolics.

• Base conversion, input such as \texttt{0xAB.CDEF}.

• Random numbers (pgfmath provides \texttt{rnd}, \texttt{rand}, \texttt{random}), with seed reset at every \texttt{\fp\_set:Nn}.

• Factorial (not with \texttt{!}), gamma function.

• Improve coefficients of the \texttt{sin} and \texttt{tan} series.

• Treat upper and lower case letters identically in identifiers, and ignore underscores.

• Add an \texttt{array(1,2,3)} and \texttt{i=complex(0,1)}.

• Provide an experimental \texttt{map} function? Perhaps easier to implement if it is a single character, \texttt{@sin(1,2)}?

• Provide \texttt{\fp\_if\_nan:NTF}, and an \texttt{isnan} function?

• Support keyword arguments?

Pgfmath also provides box-measurements (depth, height, width), but boxes are not possible expandably.

Bugs. (Exclamation points mark important bugs.)

• Check that functions are monotonic when they should.

• Add exceptions to ?:, \texttt{!<>?}, \texttt{&\&}, \texttt{||}, and \texttt{!}.

• Logarithms of numbers very close to 1 are inaccurate.

• When rounding towards \(-\infty\), \texttt{\dim\_to\_fp:n \{0pt\}} should return \(-0\), not \(+0\).

• The result of \((\pm0) + (\pm0)\), of \(x + (-x)\), and of \((-x) + x\) should depend on the rounding mode.
• 0e9999999999 gives a TeX “number too large” error.

• Subnormals are not implemented.

• The overflow trap receives the wrong argument in l3fp-expo (see \texttt{exp(1e5678)} in m3fp-traps001).

Possible optimizations/improvements.

• Document that l3trial/l3fp-types introduces tools for adding new types.

• In subsection 9.1, write a grammar.

• Fix the \texttt{\textsc{two bars}} business with the index.

• It would be nice if the \texttt{parse} auxiliaries for each operation were set up in the corresponding module, rather than centralizing in \texttt{l3fp-parse}.

• Some functions should get an \_o ending to indicate that they expand after their result.

• More care should be given to distinguish expandable/restricted expandable (auxiliary and internal) functions.

• The code for the \texttt{ternary} set of functions is ugly.

• There are many ~ missing in the doc to avoid bad line-breaks.

• The algorithm for computing the logarithm of the significand could be made to use a 5 terms Taylor series instead of 10 terms by taking \( c = \frac{2000}{(\lfloor 200x \rfloor + 1)} \in [10, 95] \) instead of \( c \in [1, 10] \). Also, it would then be possible to simplify the computation of \( t \). However, we would then have to hard-code the logarithms of 44 small integers instead of 9.

• Improve notations in the explanations of the division algorithm (l3fp-basics).

• Understand and document \texttt{\_\_fp\_basics\_pack\_weird\_low:NNNNw} and \texttt{\_\_fp\_basics\_pack\_weird\_high:NNNNNNNNNw} better. Move the other \texttt{basics\_pack} auxiliaries to \texttt{l3fp-aux} under a better name.

• Find out if underflow can really occur for trigonometric functions, and redoc as appropriate.

• Add bibliography. Some of Kahan’s articles, some previous TeX fp packages, the international standards,….

• Also take into account the “inexact” exception?

• Support multi-character prefix operators (\textit{e.g.}, @/ or whatever)? Perhaps for including comments inside the computation itself??
Part XXIII
The \texttt{l3candidates} package
Experimental additions to \texttt{l3kernel}

1 Important notice

This module provides a space in which functions can be added to \texttt{l3kernel} (\texttt{expl3}) while still being experimental.

As such, the functions here may not remain in their current form, or indeed at all, in \texttt{l3kernel} in the future.

In contrast to the material in \texttt{l3experimental}, the functions here are all \textit{small} additions to the kernel. We encourage programmers to test them out and report back on the \LaTeX-L mailing list.

Thus, if you intend to use any of these functions from the candidate module in a public package offered to others for productive use (e.g., being placed on CTAN) please consider the following points carefully:

\begin{itemize}
  \item Be prepared that your public packages might require updating when such functions are being finalized.
  \item Consider informing us that you use a particular function in your public package, e.g., by discussing this on the \LaTeX-L mailing list. This way it becomes easier to coordinate any updates necessary without a issues for the users of your package.
  \item Discussing and understanding use cases for a particular addition or concept also helps to ensure that we provide the right interfaces in the final version so please give us feedback if you consider a certain candidate function useful (or not).
\end{itemize}

We only add functions in this space if we consider them being serious candidates for a final inclusion into the kernel. However, real use sometimes leads to better ideas, so functions from this module are \textbf{not necessarily stable} and we may have to adjust them!

2 Additions to \texttt{l3basics}

\begin{verbatim}
\cs_log:N  \cs_log:c
\__kernel_register_log:N  \__kernel_register_log:c
\end{verbatim}

\texttt{\cs_log:N \texttt{(control sequence)}}

Writes the definition of the \texttt{(control sequence)} in the log file. See also \texttt{\cs_show:N} which displays the result in the terminal.

\begin{verbatim}
\__kernel_register_log:N  \__kernel_register_log:c
\end{verbatim}

\texttt{\__kernel_register_log:N \texttt{(register)}}

Used to write the contents of a \TeX{} register to the log file in a form similar to \texttt{\__kernel_register_show:N}.  

199
3 Additions to \texttt{l3box}

3.1 Affine transformations

Affine transformations are changes which (informally) preserve straight lines. Simple translations are affine transformations, but are better handled in \TeX{} by doing the translation first, then inserting an unmodified box. On the other hand, rotation and resizing of boxed material can best be handled by modifying boxes. These transformations are described here.

\begin{verbatim}
\box_resize:Nnn \box_resize:Nnn \box_resize:cnn
\box_resize_to_ht_plus_dp:Nn \box_resize_to_ht_plus_dp:cn
\end{verbatim}

Resize the \langle box \rangle to \langle x-size \rangle horizontally and \langle y-size \rangle vertically (both of the sizes are dimension expressions). The \langle y-size \rangle is the vertical size (height plus depth) of the box. The updated \langle box \rangle will be an hbox, irrespective of the nature of the \langle box \rangle before the resizing is applied. Negative sizes will cause the material in the \langle box \rangle to be reversed in direction, but the reference point of the \langle box \rangle will be unchanged. Thus negative \langle y \rangle-sizes will result in a box with depth dependent on the height of the original box a height dependent on the depth. The resizing applies within the current \TeX{} group level.

\begin{verbatim}
\box_resize_to_ht:Nn \box_resize_to_ht:cn
\end{verbatim}

Resize the \langle box \rangle to \langle y-size \rangle vertically, scaling the horizontal size by the same amount (\langle y-size \rangle is a dimension expression). The \langle y-size \rangle is the vertical size (height plus depth) of the box. The updated \langle box \rangle will be an hbox, irrespective of the nature of the \langle box \rangle before the resizing is applied. A negative size will cause the material in the \langle box \rangle to be reversed in direction, but the reference point of the \langle box \rangle will be unchanged. Thus negative y-sizes will result in a box with depth dependent on the height of the original box and height dependent on the depth of the original. The resizing applies within the current \TeX{} group level.
\texttt{\textbackslash box\_resize\_to\_wd:NN} \texttt{\textbackslash box\_resize\_to\_wd:CN}
\texttt{\box\_resize\_to\_wd:NN (box) \{x-size\}}

Resize the \texttt{\{box\}} to \texttt{\{x-size\}} horizontally, scaling the vertical size by the same amount (\texttt{\{x-size\}} is a dimension expression). The updated \texttt{\{box\}} will be an hbox, irrespective of the nature of the \texttt{\{box\}} before the resizing is applied. A negative size will cause the material in the \texttt{\{box\}} to be reversed in direction, but the reference point of the \texttt{\{box\}} will be unchanged. Thus negative \texttt{\{y-size\}} will result in a box a depth dependent on the height of the original box a height dependent on the depth. The resizing applies within the current \TeX{} group level.

\texttt{\textbackslash box\_resize\_to\_wd\_and\_ht:NN} \texttt{\textbackslash box\_resize\_to\_wd\_and\_ht:CN}
\texttt{\box\_resize\_to\_wd\_and\_ht:NN (box) \{x-size\} \{y-size\}}

Resize the \texttt{\{box\}} to a \texttt{\{height\}} of \texttt{\{x-size\}} horizontally and \texttt{\{y-size\}} vertically (both of the sizes are dimension expressions). The \texttt{\{y-size\}} is the \texttt{\{height\}} of the box, ignoring any depth. The updated \texttt{\{box\}} will be an hbox, irrespective of the nature of the \texttt{\{box\}} before the resizing is applied. Negative sizes will cause the material in the \texttt{\{box\}} to be reversed in direction, but the reference point of the \texttt{\{box\}} will be unchanged.

\texttt{\textbackslash box\_rotate:NN} \texttt{\textbackslash box\_rotate:CN}
\texttt{\box\_rotate:NN (box) \{angle\}}

Rotates the \texttt{\{box\}} by \texttt{\{angle\}} (in degrees) anti-clockwise about its reference point. The reference point of the updated box will be moved horizontally such that it is at the left side of the smallest rectangle enclosing the rotated material. The updated \texttt{\{box\}} will be an hbox, irrespective of the nature of the \texttt{\{box\}} before the rotation is applied. The rotation applies within the current \TeX{} group level.

\texttt{\textbackslash box\_scale:NN} \texttt{\textbackslash box\_scale:CN}
\texttt{\box\_scale:NN (box) \{x-scale\} \{y-scale\}}

Scales the \texttt{\{box\}} by factors \texttt{\{x-scale\}} and \texttt{\{y-scale\}} in the horizontal and vertical directions, respectively (both scales are integer expressions). The updated \texttt{\{box\}} will be an hbox, irrespective of the nature of the \texttt{\{box\}} before the scaling is applied. Negative scalings will cause the material in the \texttt{\{box\}} to be reversed in direction, but the reference point of the \texttt{\{box\}} will be unchanged. Thus negative \texttt{\{y-scale\}} will result in a box a depth dependent on the height of the original box a height dependent on the depth. The resizing applies within the current \TeX{} group level.
3.2 Viewing part of a box

\box_clip:N \box_clip:N \box_clip:c

Clips the \langle box \rangle in the output so that only material inside the bounding box is displayed in the output. The updated \langle box \rangle will be an hbox, irrespective of the nature of the \langle box \rangle before the clipping is applied. The clipping applies within the current \TeX{} group level.

These functions require the \LaTeX{} native drivers: they will not work with the \LaTeX{} 2ε graphics drivers!

\TeX{}hackers note: Clipping is implemented by the driver, and as such the full content of the box is placed in the output file. Thus clipping does not remove any information from the raw output, and hidden material can therefore be viewed by direct examination of the file.

\box_trim:Nnnnn \box_trim:Nnnnn \box_trim:cnnnn

Adjusts the bounding box of the \langle box \rangle \langle left \rangle is removed from the left-hand edge of the bounding box, \langle right \rangle from the right-hand edge and so forth. All adjustments are \langle dimension expressions \rangle. Material output of the bounding box will still be displayed in the output unless \box_clip:N is subsequently applied. The updated \langle box \rangle will be an hbox, irrespective of the nature of the \langle box \rangle before the trim operation is applied. The adjustment applies within the current \TeX{} group level. The behavior of the operation where the trims requested is greater than the size of the box is undefined.

\box_viewport:Nnnnn \box_viewport:Nnnnn \box_viewport:cnnnn

Adjusts the bounding box of the \langle box \rangle such that it has lower-left co-ordinates \langle llx \rangle, \langle lly \rangle and upper-right co-ordinates \langle urx \rangle, \langle ury \rangle. All four co-ordinate positions are \langle dimension expressions \rangle. Material output of the bounding box will still be displayed in the output unless \box_clip:N is subsequently applied. The updated \langle box \rangle will be an hbox, irrespective of the nature of the \langle box \rangle before the viewport operation is applied. The adjustment applies within the current \TeX{} group level.

3.3 Internal variables

\l__box_angle_fp

The angle through which a box is rotated by \box_rotate:Nn, given in degrees counter-clockwise. This value is required by the underlying driver code in l3driver to carry out the driver-dependent part of box rotation.

\l__box_cos_fp \l__box_sin_fp

The sine and cosine of the angle through which a box is rotated by \box_rotate:Nn: the values refer to the angle counter-clockwise. These values are required by the underlying driver code in l3driver to carry out the driver-dependent part of box rotation.
The scaling factors by which a box is scaled by \texttt{\box_scale:Nnn} or \texttt{\box_resize:Nnn}. These values are required by the underlying driver code in \texttt{l3driver} to carry out the driver-dependent part of box rotation.

Box used for affine transformations, which is used to contain rotated material when applying \texttt{\box_rotate:Nn}. This box must be correctly constructed for the driver-dependent code in \texttt{l3driver} to function correctly.

### 4 Additions to \texttt{l3clist}

\texttt{\clist_log:N} \texttt{\clist_log:N (comma list)} \texttt{\clist_log:c} \texttt{\clist_log:n \langle tokens\rangle}  

Wrote the entries in the \langle comma list\rangle in the log file. See also \texttt{\clist_show:N} which displays the result in the terminal.

### 5 Additions to \texttt{l3coffins}

\texttt{\coffin_resize:Nnn} \texttt{\coffin_resize:cnn}  

Resized the \langle coffin\rangle to \langle width\rangle and \langle total-height\rangle, both of which should be given as dimension expressions.

\texttt{\coffin_rotate:Nn} \texttt{\coffin_rotate:cn}  

Rotates the \langle coffin\rangle by the given \langle angle\rangle (given in degrees counter-clockwise). This process will rotate both the coffin content and poles. Multiple rotations will not result in the bounding box of the coffin growing unnecessarily.

\texttt{\coffin_scale:Nnn} \texttt{\coffin_scale:cnn}  

Scales the \langle coffin\rangle by a factors \langle x-scale\rangle and \langle y-scale\rangle in the horizontal and vertical directions, respectively. The two scale factors should be given as real numbers.

\texttt{\coffin_log_structure:N} \texttt{\coffin_log_structure:c}  

This function writes the structural information about the \langle coffin\rangle in the log file. The width, height and depth of the typeset material are given, along with the location of all of the poles of the coffin. See also \texttt{\coffin_show_structure:N} which displays the result in the terminal.
6 Additions to l3file

\file_if_exist_input:nTF
\file_if_exist_input:nTF \{ (file name) \} \{ (true code) \} \{ (false code) \}

Searches for \( (file \ name) \) using the current \TeX\ search path and the additional paths controlled by \file_path_include:n. If found, inserts the \( (true \ code) \) then reads in the file as additional \LaTeX\ source as described for \file_input:n. Note that \file_if_exist_input:nTF does not raise an error if the file is not found, in contrast to \file_input:n.

\ior_map_inline:Nn
\ior_map_inline:Nn \{ (stream) \} \{ (inline function) \}

Applies the \( (inline \ function) \) to \( (lines) \) obtained by reading one or more lines (until an equal number of left and right braces are found) from the \( (stream) \). The \( (inline \ function) \) should consist of code which will receive the \( (line) \) as \#1. Note that \TeX\ removes trailing space and tab characters (character codes 32 and 9) from every line upon input. \TeX\ also ignores any trailing new-line marker from the file it reads.

\ior_str_map_inline:Nn
\ior_str_map_inline:Nn \{ (stream) \} \{ (inline function) \}

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

\ior_map_break:
\ior_map_break:

Used to terminate a \ior_map... function before all lines from the \( (stream) \) have been processed. This will normally take place within a conditional statement, for example

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

Use outside of a \ior_map... scenario will lead to low level \TeX\ errors.

\TeXhackers\ note: When the mapping is broken, additional tokens may be inserted by the internal macro \__prg_break_point:Nn before further items are taken from the input stream. This will depend on the design of the mapping function.
\ior_map_break:n \ior_map_break:n \{\langle tokens\rangle\}

Used to terminate a \ior_map... function before all lines in the \langle stream\rangle have been processed, inserting the \langle tokens\rangle after the mapping has ended. This will normally take place within a conditional statement, for example

\ior_map_inline:Nn \l_my_ior
\{
    \str_if_eq:nnTF { #1 } { bingo } { \ior_map_break:n { \langle tokens\rangle } { \%
        Do something useful }
    }
\}

Use outside of a \ior_map... scenario will lead to low level \TeX errors.

\TeXhackers note: When the mapping is broken, additional tokens may be inserted by the internal macro \_\_prg_break_point:Nn before the \langle tokens\rangle are inserted into the input stream. This will depend on the design of the mapping function.

\ior_log_streams:
\iow_log_streams:

New: 2014-08-22

\fp_log:N \fp_log:N \langle fp var\rangle
\fp_log:c \fp_log:n \{\langle floating point expression\rangle\}

Evaluates the \langle floating point expression\rangle and writes the result in the log file.

\int_log:N \int_log:N \langle integer\rangle
\int_log:c \int_log:n \{\langle integer expression\rangle\}

\int_log:n \int_log:n \langle integer expression\rangle

New: 2014-08-22

\int_log:n \int_log:n \langle integer expression\rangle

Writes the result of evaluating the \langle integer expression\rangle in the log file.

7 Additions to l3fp

8 Additions to l3int
9 Additions to \texttt{l3keys}

\begin{verbatim}
\keys_log:nn \keys_log:nn \{\langle module\rangle\} \{\langle key\rangle\}
\end{verbatim}

Writes in the log file the function which is used to actually implement a \langle key\rangle for a \langle module\rangle.

10 Additions to \texttt{l3msg}

\begin{verbatim}
\__msg_log:nn \__msg_log:nn \{\langle module\rangle\} \{\langle message\rangle\} \{\langle arg one\rangle\}
\end{verbatim}

Writes the \langle message\rangle from \langle module\rangle in the log file without formatting. Used in messages which print complex variable contents completely.

\begin{verbatim}
\__msg_log_variable:Nnn \__msg_log_variable:Nnn \{\langle variable\rangle\} \{\langle type\rangle\} \{\langle formatted content\rangle\}
\end{verbatim}

Writes the \langle formatted content\rangle of the \langle variable\rangle of \langle type\rangle in the log file. The \langle formatted content\rangle will be processed as the first argument in a call to \texttt{\iow_wrap:nnn}, hence \texttt{\\, \_\_} and other formatting sequences can be used. Once expanded and processed, the \langle formatted content\rangle must either be empty or contain \texttt{>}; everything until the first \texttt{>} will be removed.

\begin{verbatim}
\__msg_log_wrap:n \__msg_log_wrap:n \{\langle formatted text\rangle\}
\end{verbatim}

Writes the \langle formatted text\rangle in the log file. After expansion, unless it is empty, the \langle formatted text\rangle must contain \texttt{>}, and the part of \langle formatted text\rangle before the first \texttt{>} is removed. Failure to do so causes low-level \TeX{} errors.

\begin{verbatim}
\__msg_log_value:n \__msg_log_value:x \__msg_log_value:x \{\langle tokens\rangle\}
\end{verbatim}

Writes \texttt{\langle tokens\rangle} in the log file.

11 Additions to \texttt{l3prg}

\begin{verbatim}
\bool_log:N \bool_log:N \{\langle boolean\rangle\}
\end{verbatim}

Writes the logical truth of the \langle boolean\rangle in the log file.

\begin{verbatim}
\bool_log:n \bool_log:n \{\langle boolean expression\rangle\}
\end{verbatim}

Writes the logical truth of the \langle boolean expression\rangle in the log file.
12 Additions to \texttt{l3prop}

\texttt{\prop_map_tokens:Nn} \hspace{1em} \texttt{\prop_map_tokens:cn}

Analogue of \texttt{\prop_map_function:NN} which maps several tokens instead of a single function. The \texttt{\langle code\rangle} receives each key–value pair in the \texttt{\langle property list\rangle} as two trailing brace groups. For instance,

\begin{verbatim}
\prop_map_tokens:Nn \l_my_prop \{ \str_if_eq:nnT { mykey } \}
\end{verbatim}

will expand to the value corresponding to \texttt{mykey}: for each pair in \texttt{\l_my_prop} the function \texttt{\str_if_eq:nnT} receives \texttt{mykey}, the \texttt{\langle key\rangle} and the \texttt{\langle value\rangle} as its three arguments. For that specific task, \texttt{\prop_item:Nn} is faster.

\texttt{\prop_log:N} \hspace{1em} \texttt{\prop_log:c}

Writes the entries in the \texttt{\langle property list\rangle} in the log file.

13 Additions to \texttt{l3seq}

\texttt{\seq_mapthread_function:NNN} \hspace{1em} \texttt{\seq_mapthread_function:NNN} \hspace{1em} \texttt{\seq_mapthread_function:(NcN|cNN|ccN)}

Applies \texttt{\langle function\rangle} to every pair of items \texttt{\langle seq1\text{-item}\rangle}–\texttt{\langle seq2\text{-item}\rangle} from the two sequences, returning items from both sequences from left to right. The \texttt{\langle function\rangle} will receive two \texttt{n-type} arguments for each iteration. The mapping will terminate when the end of either sequence is reached (\textit{i.e.} whichever sequence has fewer items determines how many iterations occur).

\texttt{\seq_set_filter:NNn} \hspace{1em} \texttt{\seq_gset_filter:NNn}

Evaluates the \texttt{\langle inline boolean\rangle} for every \texttt{\langle item\rangle} stored within the \texttt{\langle sequence2\rangle}. The \texttt{\langle inline boolean\rangle} will receive the \texttt{\langle item\rangle} as \#1. The sequence of all \texttt{\langle items\rangle} for which the \texttt{\langle inline boolean\rangle} evaluated to \texttt{true} is assigned to \texttt{\langle sequence1\rangle}.

\textbf{\texttt{\TeX}hackers note:} Contrarily to other mapping functions, \texttt{\seq_map_break:} cannot be used in this function, and will lead to low-level \texttt{\TeX} errors.
\seq_set_map:NNn \sequence_1 \sequence_2 \inline_function

Applies \inline_function to every \item stored within the \sequence_2. The \inline_function should consist of code which will receive the \item as #1. The sequence resulting from x-expanding \inline_function applied to each \item is assigned to \sequence_1. As such, the code in \inline_function should be expandable.

\TeXhackers note: Contrarily to other mapping functions, \seq_map_break: cannot be used in this function, and will lead to low-level \TeX errors.

\seq_log:N \sequence
\seq_log:c
New: 2014-08-12

Writes the entries in the \sequence in the log file.

14 Additions to l3skip

\skip_split_finite_else_action:nnNN \skip_split_finite_else_action:nnNN \skipexpr \action \dimen_1 \dimen_2

Checks if the \skipexpr contains finite glue. If it does then it assigns \dimen_1 the stretch component and \dimen_2 the shrink component. If it contains infinite glue set \dimen_1 and \dimen_2 to 0 pt and place #2 into the input stream: this is usually an error or warning message of some sort.

\dim_log:N \dim_log:c
New: 2014-08-22

Writes the value of the \dimension in the log file.

\dim_log:n
New: 2014-08-22

Writes the result of evaluating the \dimension expression in the log file.

\skip_log:N \skip_log:c
New: 2014-08-22

Writes the value of the \skip in the log file.

\skip_log:n
New: 2014-08-22

Writes the result of evaluating the \skip expression in the log file.

\muskip_log:N \muskip_log:c
New: 2014-08-22

Writes the value of the \muskip in the log file.
\muskip_log:n \{\muskip expression\}

Writes the result of evaluating the \textit{\muskip expression} in the log file.

15 Additions to \texttt{l3tl}

\tl_if_single_token_p:n \{\texttt{token list}\}
\tl_if_single_token:nTF \{\texttt{token list}\} \{\texttt{true code}\} \{\texttt{false code}\}

Tests if the token list consists of exactly one token, \textit{i.e.} is either a single space character or a single “normal” token. Token groups (\{\ldots\}) are not single tokens.

\tl_reverse_tokens:n \{\texttt{tokens}\}

This function, which works directly on TeX tokens, reverses the order of the \texttt{tokens}: the first will be the last and the last will become first. Spaces are preserved. The reversal also operates within brace groups, but the braces themselves are not exchanged, as this would lead to an unbalanced token list. For instance, \texttt{\tl_reverse_tokens:n \{a\{b()\}\}} leaves \{\)(b\}~a in the input stream. This function requires two steps of expansion.

\textbf{TeXhackers note:} The result is returned within the \texttt{unexpanded} primitive (\texttt{\exp_not:n}), which means that the token list will not expand further when appearing in an \texttt{x}-type argument expansion.

\tl_count_tokens:n \{\texttt{tokens}\}

Counts the number of TeX tokens in the \texttt{tokens} and leaves this information in the input stream. Every token, including spaces and braces, contributes one to the total; thus for instance, the token count of \texttt{a\{bc\}} is 6. This function requires three expansions, giving an \texttt{integer denotation}.

\tl_expandable_uppercase:n \{\texttt{tokens}\}
\tl_expandable_lowercase:n \{\texttt{tokens}\}

The \texttt{\tl_expandable_uppercase:n} function works through all of the \texttt{tokens}, replacing characters in the range \texttt{a\textendash z} (with arbitrary category code) by the corresponding letter in the range \texttt{A\textendash Z}, with category code 11 (letter). Similarly, \texttt{\tl_expandable_lowercase:n} replaces characters in the range \texttt{A\textendash Z} by letters in the range \texttt{a\textendash z}, and leaves other tokens unchanged. This function requires two steps of expansion.

\textbf{TeXhackers note:} Begin-group and end-group characters are normalized and become \{ and \}, respectively. The result is returned within the \texttt{unexpanded} primitive (\texttt{\exp_not:n}), which means that the token list will not expand further when appearing in an \texttt{x}-type argument expansion.
These functions are intended to be applied to input which may be regarded broadly as "text". They traverse the \textit{tokens} and change the case of characters as discussed below. The character code of the characters replaced may be arbitrary: the replacement characters will have standard document-level category codes (11 for letters, 12 for letter-like characters which can also be case-changed). Begin-group and end-group characters in the \textit{tokens} are normalized and become \{ and \}, respectively.

Importantly, notice that these functions are intended for working with user text for typesetting. For case changing programmatic data see the \texttt{l3str} module and discussion there of \texttt{\str_lower_case:n}, \texttt{\str_upper_case:n} and \texttt{\str_fold_case:n}.

The functions perform expansion on the input in most cases. In particular, input in the form of token lists or expandable functions will be expanded \textit{unless} it falls within one of the special handling classes described below. This expansion approach means that in general the result of case changing will match the "natural" outcome expected from a "functional" approach to case modification. For example

\begin{verbatim}
\tl_set:Nn \l_tmpa_tl { hello }
\tl_upper_case:n { \l_tmpa_tl \c_space_tl world }
\end{verbatim}

will produce

\texttt{HELLO WORLD}

The expansion approach taken means that in package mode any L\LaTeX\ “robust” commands which may appear in the input should be converted to engine-protected versions using for example the \texttt{\robustify} command from the \texttt{etoolbox} package.

Case changing will not take place within math mode material so for example

\begin{verbatim}
\tl_upper_case:n { Some-text-$y = mx + c$-with-\{Braces} }
\end{verbatim}

will become

\texttt{SOME TEXT $y = mx + c$ WITH \{BRACES}\}

Material inside math mode is left entirely unchanged: in particular, no expansion is undertaken.

Detection of math mode is controlled by the list of tokens in \texttt{\l_tl_case_change_math_tl}, which should be in open–close pairs. In package mode the standard settings is

\begin{verbatim}
\$ $ \left( \right)
\end{verbatim}
Case changing can be prevented by using any command on the list \texttt{\l_tl_case_change_exclude_tl}. Each entry should be a function to be followed by one argument: the latter will be preserved as-is with no expansion. Thus for example following

\begin{verbatim}
\tl_put_right:Nn \l_tl_case_change_exclude_tl { \NoChangeCase }
\end{verbatim}

the input

\begin{verbatim}
\tl_upper_case:n
{ Some~text-$y = mx + c$~with-\NoChangeCase {Protection} }
\end{verbatim}

will result in

\begin{verbatim}
SOME TEXT $y = mx + c$ WITH \NoChangeCase {Protection}
\end{verbatim}

Notice that the case changing mapping preserves the inclusion of the escape functions: it is left to other code to provide suitable definitions (typically equivalent to \texttt{\use:n}). In particular, the result of case changing is returned protected by \texttt{\exp_not:n}.

When used with L\TeX{} the commands \cite, \ensuremath, \label and \ref are automatically included in the list for exclusion from case changing.

In package mode, the case change system will also convert text stored using the L\TeX{} “LICR” approach. This will upper/lower case tokens as implemented for the font encodings T1, T2, T5 and LGR (see the behaviour of the L\TeX{} command \texttt{\MakeUppercase}). Note that these commands will automatically be protected from expansion.

“Mixed” case conversion may be regarded informally as converting the first character of the \texttt{(tokens)} to upper case and the rest to lower case. However, the process is more complex than this as there are some situations where a single lower case character maps to a special form, for example \texttt{ij} in Dutch which becomes \texttt{IJ}. As such, \texttt{\tl_mixed_case:n(n)} implement a more sophisticated mapping which accounts for this and for modifying accents on the first letter. Spaces at the start of the \texttt{(tokens)} are ignored when finding the first “letter” for conversion.

\begin{verbatim}
\tl_mixed_case:n { hello-WORLD } % => "Hello world"
\tl_mixed_case:n { ~hello-WORLD } % => " Hello world"
\tl_mixed_case:n { \{hello\}-WORLD } % => "\{Hello\} world"
\end{verbatim}

When finding the first “letter” for this process, any content in math mode or covered by \texttt{\l_tl_case_change_exclude_tl} is ignored.

(Note that the Unicode Consortium describe this as “title case”, but that in English title case applies on a word-by-word basis. The “mixed” case implemented here is a lower level concept needed for both “title” and “sentence” casing of text.)
The list of characters to ignore when searching for the first “letter” in mixed-casing is
determined by \_tl\_mixed\_change\_ignore\_tl. This has the standard setting

\[ \{ \{ ' - \} \} \]

where comparisons are made on a character basis.

As is generally true for expl3, these functions are designed to work with Unicode
input only. As such, when used with pdf\TeX\ only the characters a–zA–Z are modified.
When used with Xe\TeX\ or Lua\TeX\ a full range of Unicode transformations are enabled.
Specifically, the standard mappings here follow those defined by the Unicode Consortium
in UnicodeData.txt and SpecialCasing.txt. Note that in some cases, pdf\TeX\ can
interpret the input to a case change but not generate the correct output (for example in
the mapping i to I-dot in Turkish): in these cases the input is left unchanged.

Context-sensitive mappings are enabled: language-dependent cases are discussed
below. All context detection works before any expansion of following tokens and thus
any control sequences are taken as the end of the current “word”/character context.

The “final sigma” rule for Greek letters is enabled and active for all inputs. It is imple-
mented here in a modified form which takes account of the requirements of the likely real
use cases, performance and expandability. A capital sigma will map to a final-sigma if
it is followed by a space, the end of the input or one of the characters listed in \_tl\_-
change\_case\_after\_final\_sigma\_tl, which has standard setting

\[ ] \} . ; , ! ? ' " \]

with comparisons are made on a character basis.

Language-sensitive conversions are enabled using the \langle language\rangle argument, and fol-
low Unicode Consortium guidelines. Currently, the languages recognised for special han-
dling are as follows.

- Azeri and Turkish (az and tr). The case pairs I/i-dotless and I-dot/i are activated
  for these languages. The combining dot mark is removed when lower casing I-dot
  and introduced when upper casing i-dotless.

- Lithuanian (lt). The lower case letters i and j should retain a dot above when the
  accents grave, acute or tilde are present. This is implemented for lower casing of
  the relevant upper case letters both when input as single Unicode codepoints and
  when using combining accents. The combining dot is removed when upper casing
  in these cases. Note that only the accents used in Lithuanian are covered: the
  behaviour of other accents are not modified.

- Dutch (nl). Capitalisation of ij at the beginning of mixed cased input produces IJ
  rather than Ij. The output retains two separate letters, thus this transformation
  is available using pdf\TeX.
Creating additional context-sensitive mappings requires knowledge of the underlying mapping implementation used here. The team are happy to add these to the kernel where they are well-documented (e.g. in Unicode Consortium or relevant government publications).

\tl_set_from_file:Nnn \tl_set_from_file:cn \tl_gset_from_file:Nnn \tl_gset_from_file:cn

New: 2014-06-25

\tl_set_from_file_x:Nnn \tl_set_from_file_x:cn \tl_gset_from_file_x:Nnn \tl_gset_from_file_x:cn

New: 2014-06-25

\tl_set_from_file:nn \tl_set_from_file:cnn \tl_gset_from_file:nn \tl_gset_from_file:cn

New: 2014-06-25

\tl_log:N \tl_log:C

New: 2014-08-22

\tl_log:N \tl_log:c

New: 2014-08-22

\tl_log:n \tl_log:n

New: 2014-08-22

16 Additions to l3tokens

\char_set_active:Npn \char_set_active:Npx

\char_gset_active:Npn \char_gset_active:Npx

Makes \langle char \rangle an active character to expand to \langle code \rangle as replacement text. Within the \langle code \rangle, the \langle parameters \rangle (#1, #2, etc.) will be replaced by those absorbed. The \langle char \rangle is made active within the current \TeX\ group level, and the definition is also local.

Makes \langle char \rangle an active character to expand to \langle code \rangle as replacement text. Within the \langle code \rangle, the \langle parameters \rangle (#1, #2, etc.) will be replaced by those absorbed. The \langle char \rangle is made active within the current \TeX\ group level, but the definition is global. This function is therefore suited to cases where an active character definition should be applied only in some context (where the \langle char \rangle is again made active).
\char_set_active_eq:NN \char_set_active_eq:NN \langle \text{char} \rangle (\text{function})

Makes \langle \text{char} \rangle an active character equivalent in meaning to the \langle \text{function} \rangle (which may itself be an active character). The \langle \text{char} \rangle is made active within the current \LaTeX{} group level, and the definition is also local.

\char_gset_active_eq:NN \char_gset_active_eq:NN \langle \text{char} \rangle \langle \text{function} \rangle

Makes \langle \text{char} \rangle an active character equivalent in meaning to the \langle \text{function} \rangle (which may itself be an active character). The \langle \text{char} \rangle is made active within the current \LaTeX{} group level, but the definition is global. This function is therefore suited to cases where an active character definition should be applied only in some context (where the \langle \text{char} \rangle is again made active).

\peek_N_type:TF \peek_N_type:TF \{\langle \text{true code} \rangle\} \{\langle \text{false code} \rangle\}

Tests if the next \langle \text{token} \rangle in the input stream can be safely grabbed as an \textit{N}-type argument. The test will be \langle \text{false} \rangle if the next \langle \text{token} \rangle is either an explicit or implicit begin-group or end-group token (with any character code), or an explicit or implicit space character (with character code 32 and category code 10), or an outer token (never used in \LaTeX{}3) and \langle \text{true} \rangle in all other cases. Note that a \langle \text{true} \rangle result ensures that the next \langle \text{token} \rangle is a valid \textit{N}-type argument. However, if the next \langle \text{token} \rangle is for instance \texttt{\c_space_token}, the test will take the \langle \text{false} \rangle branch, even though the next \langle \text{token} \rangle is in fact a valid \textit{N}-type argument. The \langle \text{token} \rangle will be left in the input stream after the \langle \text{true code} \rangle or \langle \text{false code} \rangle (as appropriate to the result of the test).
Part XXIV

The l3drivers package

Drivers

\[ \text{l3}\text{\LaTeX} \] relies on drivers in order to carry out a number of tasks, such as using color, including graphics and setting up hyper-links. The nature of the code required depends on the exact driver in use. Currently, \[ \text{l3}\text{\LaTeX}3 \] is aware of the following drivers:

- **pdfmode**: The “driver” for direct PDF output by \emph{both} pdf\LaTeX{} and Lua\LaTeX{} (no separate driver is used in this case: the engine deals with PDF creation itself).

- **dvips**: The dvips program, which works in conjunction with pdf\LaTeX{} or Lua\LaTeX{} in DVI mode.

- **dvipdfmx**: The dvipdfmx program, which works in conjunction with pdf\LaTeX{} or Lua\LaTeX{} in DVI mode.

- **xdvipdfmx**: The driver used by X\LaTeX{}.

The code here is all very low-level, and should not in general be used outside of the kernel. It is also important to note that many of the functions here are closely tied to the immediate level “up”: several variable values must be in the correct locations for the driver code to function.

1 Box clipping

\[ \_\_driver\_box\_use\_clip:N \] \_\_driver\_box\_use\_clip:N \text{ (box)}

Inserts the content of the \text{ (box)} at the current insertion point such that any material outside of the bounding box will not be displayed by the driver. The material in the \text{ (box)} is still placed in the output stream: the clipping takes place at a driver level.

This function should only be used within a surrounding horizontal box construct.
2 Box rotation and scaling

\_\_driver_box_rotate_begin: \_\_driver_box_rotate_begin:
\_\_driver_box_rotate_end: box_use:N \l__box_internal_box
\_\_driver_box_rotate_end:

Rotates the \textit{box material} anti-clockwise around the current insertion point. The angle of rotation (in degrees counter-clockwise) and the sine and cosine of this angle should be stored in \l__box_angle_fp, \l__box_sin_fp and \l__box_cos_fp, respectively. Typically, the box material inserted between the beginning and end markers will be stored in \l__box_internal_box: this fact is required by some drivers to obtain the correct output.

\_\_driver_box_scale_begin: \_\_driver_box_scale_begin:
\_\_driver_box_scale_end: \_\_driver_box_scale_end:

Scales the \textit{box material} (which should be either a \texttt{box_use:N} or \texttt{hbox:n} construct). The \textit{box material} is scaled by the values stored in \l__box_scale_x_fp and \l__box_scale_y_fp in the horizontal and vertical directions, respectively. This function is also reused when resizing boxes: at a driver level, only scalings are supported and so the higher-level code must convert the absolute sizes to scale factors.

3 Color support

\_\_driver_color_ensure_current: \_\_driver_color_ensure_current:

Ensures that the color used to typeset material is that which was set when the material was placed in a box. This function is therefore required inside any “color safe” box to ensure that the box may be inserted in a location where the foreground color has been altered, while preserving the color used in the box.
# Index

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

<table>
<thead>
<tr>
<th>Symbols</th>
<th>( \text{acscd} )</th>
<th>( \text{alignment commands:} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( kk )</td>
<td>191</td>
<td>( \backslash \text{c}_\text{alignment}_\text{token} )</td>
</tr>
<tr>
<td>( * )</td>
<td>192</td>
<td>( \backslash \text{Argument}_\text{One} )</td>
</tr>
<tr>
<td>( ** )</td>
<td>192</td>
<td>( \text{asec} )</td>
</tr>
<tr>
<td>( + )</td>
<td>192</td>
<td>( \text{asecd} )</td>
</tr>
<tr>
<td>( - )</td>
<td>192, 192</td>
<td>( \text{asin} )</td>
</tr>
<tr>
<td>( / )</td>
<td>192</td>
<td>( \text{asind} )</td>
</tr>
<tr>
<td>( \text{:N} )</td>
<td>35</td>
<td>( \text{atan} )</td>
</tr>
<tr>
<td>( \text{V} )</td>
<td>35</td>
<td>( \text{atan} )</td>
</tr>
<tr>
<td>( \text{:c} )</td>
<td>35</td>
<td>( \text{V} )</td>
</tr>
<tr>
<td>( \text{:f} )</td>
<td>35</td>
<td>( \text{N} )</td>
</tr>
<tr>
<td>( \text{n} )</td>
<td>35</td>
<td>( \text{gset_eq_NN} )</td>
</tr>
<tr>
<td>( \text{:o} )</td>
<td>35</td>
<td>( \text{gset_false_N} )</td>
</tr>
<tr>
<td>( \text{:p} )</td>
<td>35</td>
<td>( \text{gset_inverse_c} )</td>
</tr>
<tr>
<td>( \text{:v} )</td>
<td>35</td>
<td>( \text{gset_inverse_N} )</td>
</tr>
<tr>
<td>( \text{:x} )</td>
<td>35</td>
<td>( \text{gset_true_N} )</td>
</tr>
<tr>
<td>( \text{&lt;} )</td>
<td>192</td>
<td>( \backslash \text{bool_do_until_Nn} )</td>
</tr>
<tr>
<td>( \text{=} )</td>
<td>192</td>
<td>( \backslash \text{bool_do_until_nn} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \backslash \text{bool_do_while_Nn} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \backslash \text{bool_do_while_nn} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_gset_c} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_gset_N} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_gset_NN} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_gset_false_N} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_gset_inverse_c} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_gset_inverse_N} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_if_n(TF)} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_if_nTF} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_if_nTF} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_if_exist_nTF} )</td>
</tr>
<tr>
<td>( \text{=n:} )</td>
<td>174</td>
<td>( \text{bool_if_exist_nTF} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_if_exist_p_N} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_if_p_N} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_if_p_P} )</td>
</tr>
<tr>
<td>( \text{=n:} )</td>
<td>174</td>
<td>( \text{bool_if_p_N} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_log_N} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_new_N} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_not_P_n} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_set_c} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_set_N} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_set_P_n} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_set_eq_NN} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_set_false_N} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_set_inverse_c} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_set_inverse_N} )</td>
</tr>
<tr>
<td>( \text{?} )</td>
<td>192</td>
<td>( \text{bool_set_true_N} )</td>
</tr>
</tbody>
</table>
\clist_const:Nn
\clist_(g)clear:N
\clist\char_value_uccode:n
\char_value_sfcode:n
\char_value_lccode:n
\char_value_catcode:n
\char_set_catcode_math_toggle:N
\char_set_catcode_math_superscript:n
\char_set_catcode_math_subscript:n
\char_set_mathcode:nn
\char_set_lcode:nn
\char_set_lccode:nn
\char_set_catcode_space:n
\char_set_catcode_space:N
\char_set_catcode_parameter:n
\char_set_catcode_parameter:N
\char_set_catcode_other:n
\char_set_catcode_other:N
\char_set_catcode_math_toggle:n
\clist_count:n
\clist_count:N
\clist_get:NN
\clist_gconcat:NNN
\clist_gclear_new:N
\clist_gclear:N
\clist_gpush:Nn
\clist_gpop:NNTF
\clist_gpop:NN
\clist_get:NNTF
\clist_gremove_duplicates:N
\clist_gremove_all:Nn
\clist_gput_right:Nn
\clist_gput_left:Nn
\clist_if_in:nnTF
\clist_if_in:NnTF
\clist_if_exist_p:N
\clist_if_exist:NTF
\clist_if_empty_p:n
\clist_if_empty_p:N
\clist_if_empty:nTF
\clist_if_empty:NTF
\clist_gset:Nn
\clist_gset:N
\clist_gset:c
\clist_log:N
\clist_map_variable:nNn
\clist_map_inline:nn
\clist_map_function:NN
\clist_map_break:n
\clist_map_break:
\clist_map_...
\clist_pop:NN
\clist_new:N
\clist_map_variable:NNn
\clist_map_inline:Nn
\clist_map_function:nN
\clist_remove_duplicates:N
\clist_remove_all:Nn
\clist_put_right:Nn
\clist_put_left:Nn
\clist_push:Nn
\clist_new:N
\clist_remove_duplicates:N
\clist_remove_all:Nn
\clist_put_right:Nn
\clist_put_left:Nn
\clist_push:Nn
\clist_new:N
\clist_count:N ....... 125, 125, 128
\clist_count:n ........ 125
\clist_gclear:N .......... 120
\clist_gclear_new:N ..... 121
\clist_get:NN .......... 127, 127
\clist_get:NTF ........ 127, 127
\clist_gpop:NN ........ 127, 127
\clist_gpop:NTTF ....... 127, 127
\clist_gpush:N .......... 128
\clist_gpush:Nn .......... 128
\clist_gput_left:Nn ....... 122
\clist_gput_right:Nn ..... 122
\clist_gremove_all:Nn ... 122
\clist_gremove_duplicates:N .... 122
\clist_gset:c .......... 162
\clist_gset:N .......... 162
\clist_gset:Nn ........ 121
\clist_gset_eq:NNN ....... 121
\clist_gset_from_seq:NNN ... 121
\clist_gset_if_empty:NTF .... 123
\clist_gset_if_empty:NTF ... 123
\clist_gset_if_empty:p:N ...... 123
\clist_gset_if_empty:p:n .... 123
\clist_gset_if_exist:NTF ...... 121, 121
\clist_gset_if_exist:p:N ... 121, 121
\clist_gset_if_exist:p:n ... 121, 121
\clist_gset_if_exist_if:n ... 123
\clist_gset_if_exist_if:nT ... 123
\clist_gset_if_empty_if:nT ... 123
\clist_gset_if_empty_if:nTF . 123
\clist_gset_item:NN ...... 128, 128
\clist_gset_item:NNn .... 128, 128
\clist_gset_item:n .. 203, 203
\clist_gset_item:nN .. 203, 203
\clist_gset_map: ... 125, 125, 125, 125
\clist_gset_map_break: .... 125, 125
\clist_gset_map_break:n ... 125, 125
\clist_gset_map_function:NN ........ 45, 120, 124, 124, 124
\clist_gset_map_function:n ........ 124
\clist_gset_map_inline:Nn .. 124, 124, 124
\clist_gset_map_variable:Nn .... 124, 124
\clist_gset_map_variable:nNn ... 124, 124
\clist_gset_new:Nn ..... 120, 120, 121
\clist_gset_pop:NN ...... 127, 127
\clist_gset_pop:NTTF ...... 127, 127
\clist_gset_push:N .......... 128, 128
\clist_gset_put_left:Nn ...... 122, 122
\clist_gset_put_right:Nn ..... 122, 122
\clist_gset_remove_all:Nn .... 122, 122
\clist_gset_remove_duplicates:N ... 122, 122
\clist_gset_item:NN ...... 128, 128
\clist_gset_item:NNn .... 128, 128
\clist_gset_item:n .. 203, 203
\clist_gset_item:nN .. 203, 203
\clist_gset_map: ... 125, 125, 125, 125
\clist_gset_map_break: .... 125, 125
\clist_gset_map_break:n ... 125, 125
\clist_gset_map_function:NN ........ 45, 120, 124, 124, 124
\clist_gset_map_function:n ........ 124
\clist_gset_map_inline:Nn .. 124, 124, 124
\clist_gset_map_variable:Nn .... 124, 124
\clist_gset_map_variable:nNn ... 124, 124
\clist_gset_new:Nn ..... 120, 120, 121
\clist_gset_pop:NN ...... 127, 127
\clist_gset_pop:NTTF ...... 127, 127
\clist_gset_push:N .......... 128, 128
\clist_gset_put_left:Nn ...... 122, 122
\clist_gset_put_right:Nn ..... 122, 122
\clist_gset_remove_all:Nn .... 122, 122
\clist_gset_remove_duplicates:N ... 122, 122
\clist_gset_item:NN ...... 128, 128
\clist_gset_item:NNn .... 128, 128
\clist_gset_item:n .. 203, 203
\clist_gset_item:nN .. 203, 203
\clist_gset_map: ... 125, 125, 125, 125
\clist_gset_map_break: .... 125, 125
\clist_gset_map_break:n ... 125, 125
\clist_gset_map_function:NN ........ 45, 120, 124, 124, 124
\clist_gset_map_function:n ........ 124
\clist_gset_map_inline:Nn .. 124, 124, 124
\clist_gset_map_variable:Nn .... 124, 124
\clist_gset_map_variable:nNn ... 124, 124
\clist_gset_new:Nn ..... 120, 120, 121
\clist_gset_pop:NN ...... 127, 127
\clist_gset_pop:NTTF ...... 127, 127
\clist_gset_push:N .......... 128, 128
\clist_gset_put_left:Nn ...... 122, 122
\clist_gset_put_right:Nn ..... 122, 122
\clist_gset_remove_all:Nn .... 122, 122
\clist_gset_remove_duplicates:N ... 122, 122
\clist_gset_item:NN ...... 128, 128
\clist_gset_item:NNn .... 128, 128
\clist_gset_item:n .. 203, 203
\clist_gset_item:nN .. 203, 203
\clist_gset_map: ... 125, 125, 125, 125
\clist_gset_map_break: .... 125, 125
\clist_gset_map_break:n ... 125, 125
\clist_gset_map_function:NN ........ 45, 120, 124, 124, 124
\clist_gset_map_function:n ........ 124
\clist_gset_map_inline:Nn .. 124, 124, 124
\clist_gset_map_variable:Nn .... 124, 124
\clist_gset_map_variable:nNn ... 124, 124
\clist_gset_new:Nn ..... 120, 120, 121
\clist_gset_pop:NN ...... 127, 127
\clist_gset_pop:NTTF ...... 127, 127
\clist_gset_push:N .......... 128, 128
\clist_gset_put_left:Nn ...... 122, 122
\clist_gset_put_right:Nn ..... 122, 122
\clist_gset_remove_all:Nn .... 122, 122
\clist_gset_remove_duplicates:N ... 122, 122

219
\texttt{\textbackslash cs\_set\_nopar}\texttt{:Npn} \ldots \texttt{11, 13, 15, 55}
\texttt{\textbackslash cs\_set\_nopar}\texttt{:Npx} \ldots \texttt{13}
\texttt{\textbackslash cs\_set\_protected}\texttt{:Nn} \ldots \texttt{15, 15}
\texttt{\textbackslash cs\_set\_protected}\texttt{:Npn} \ldots \texttt{11, 13, 15}
\texttt{\textbackslash cs\_set\_protected}\texttt{:Npx} \ldots \texttt{13}
\texttt{\textbackslash cs\_set\_protected\_nopar}\texttt{:Nn} \ldots \texttt{15, 15}
\texttt{\textbackslash cs\_set\_protected\_nopar}\texttt{:Npn} \ldots \texttt{13, 13}
\texttt{\textbackslash cs\_set\_protected\_nopar}\texttt{:Npx} \ldots \texttt{13}
\texttt{\textbackslash cs\_show}\texttt{:N} \ldots \texttt{17, 17, 23, 199}
\texttt{\_\_cs\_split\_function}\texttt{:NN} \ldots \texttt{25, 25}
\texttt{\_\_cs\_tmp}\texttt{:w} \ldots \texttt{25}
\texttt{\textbackslash cs\_to\_str}\texttt{:N} \ldots \texttt{4, 4, 19, 19, 100, 106}
\texttt{\textbackslash cs\_undefined}\texttt{:N} \ldots \texttt{17, 17}
csc \ldots \texttt{193}
cscd \ldots \texttt{194}

d
dd \ldots \texttt{196}
default commands:
\texttt{\_default}\texttt{:n} \ldots \texttt{163}
\texttt{\_default}\texttt{:o} \ldots \texttt{163}
\texttt{\_default}\texttt{:V} \ldots \texttt{163}
\texttt{\_default}\texttt{:x} \ldots \texttt{163}
deg \ldots \texttt{195}
dim commands:
\texttt{\_dim\_gzero}\texttt{:N} \ldots \texttt{77}
\texttt{\_dim\_abs}\texttt{:n} \ldots \texttt{78, 78}
\texttt{\_dim\_add}\texttt{:Nn} \ldots \texttt{78, 78}
\texttt{\_dim\_case}\texttt{:nn} \ldots \texttt{81}
\texttt{\_dim\_case}\texttt{:mTF} \ldots \texttt{81, 81}
\texttt{\_dim\_compare}\texttt{:n(TP)} \ldots \texttt{76}
\texttt{\_dim\_compare}\texttt{:nHnTF} \texttt{79, 79, 81, 81, 82, 82}
\texttt{\_dim\_compare}\texttt{:nTF} \ldots \texttt{80, 80, 82, 82, 82, 82, 86}
\texttt{\_dim\_compare}\texttt{:p:n} \ldots \texttt{80, 80}
\texttt{\_dim\_compare}\texttt{:p:nHn} \texttt{79, 79}
\texttt{\_dim\_const}\texttt{:Nn} \ldots \texttt{77, 77}
\texttt{\_dim\_do\_until}\texttt{:nn} \ldots \texttt{82, 82}
\texttt{\_dim\_do\_until}\texttt{:nHn} \ldots \texttt{81, 81}
\texttt{\_dim\_do\_while}\texttt{:nn} \ldots \texttt{82, 82}
\texttt{\_dim\_do\_while}\texttt{:nHn} \ldots \texttt{81, 81}
\texttt{\_dim\_eval}\texttt{:n} \ldots \texttt{79, 80, 82, 82, 82, 83, 91}
\texttt{\_dim\_eval}\texttt{:w} \ldots \texttt{91, 91}
\texttt{\_dim\_eval}\texttt{:end} \ldots \texttt{91, 91, 91, 91}
\texttt{\_dim\_gadd}\texttt{:Nn} \ldots \texttt{78}
\texttt{\_dim\_gset}\texttt{:c} \ldots \texttt{163}
\texttt{\_dim\_gset}\texttt{:N} \ldots \texttt{163}
\texttt{\_dim\_gset}\texttt{:Nn} \ldots \texttt{78}
\texttt{\_dim\_gset}\texttt{:eq\_NN} \ldots \texttt{78}
\texttt{\_dim\_gsub}\texttt{:Nn} \ldots \texttt{78}
\texttt{\_dim\_gzero}\texttt{:N} \ldots \texttt{77}
\texttt{\_dim\_gzero\_new}\texttt{:N} \ldots \texttt{77}
\texttt{\_dim\_if\_exist}\texttt{:NTF} \ldots \texttt{77, 77}
\texttt{\_dim\_if\_exist}\texttt{:p\_N} \ldots \texttt{77, 77}
\texttt{\_dim\_log}\texttt{:N} \ldots \texttt{208, 208}
\texttt{\_dim\_log}\texttt{:n} \ldots \texttt{208, 208}
\texttt{\_dim\_max}\texttt{:nn} \ldots \texttt{78, 78}
\texttt{\_dim\_min}\texttt{:nn} \ldots \texttt{78, 78}
\texttt{\_dim\_new}\texttt{:N} \ldots \texttt{77, 77, 77}
\texttt{\_dim\_ratio}\texttt{:nn} \ldots \texttt{79, 79, 79}
\texttt{\_dim\_set}\texttt{:c} \ldots \texttt{163}
\texttt{\_dim\_set}\texttt{:N} \ldots \texttt{163}
\texttt{\_dim\_set}\texttt{:Nn} \ldots \texttt{78, 78}
\texttt{\_dim\_set}\texttt{:eq\_NN} \ldots \texttt{78, 78}
\texttt{\_dim\_show}\texttt{:N} \ldots \texttt{84, 84}
\texttt{\_dim\_show}\texttt{:n} \ldots \texttt{84, 84}
\texttt{\_dim\_sub}\texttt{:Nn} \ldots \texttt{84, 84}
\texttt{\_dim\_to\_decimal}\texttt{:n} \ldots \texttt{83, 83}
\texttt{\_dim\_to\_decimal}\texttt{:in\_BP\_n} \ldots \texttt{83, 83}
\texttt{\_dim\_to\_decimal}\texttt{:in\_UNIT\_nn} \ldots \texttt{83, 83}
\texttt{\_dim\_to\_fp}\texttt{:n} \ldots \texttt{84, 84}
\texttt{\_dim\_until\_do}\texttt{:nn} \ldots \texttt{82, 82}
\texttt{\_dim\_until\_do}\texttt{:nHn} \ldots \texttt{82, 82}
\texttt{\_dim\_use}\texttt{:N} \texttt{82, 83, 83, 83}
\texttt{\_dim\_while\_do}\texttt{:nn} \ldots \texttt{82, 82}
\texttt{\_dim\_while\_do}\texttt{:nHn} \ldots \texttt{82, 82}
\texttt{\_dim\_zero}\texttt{:N} \ldots \texttt{77, 77}
\texttt{\_dim\_zero\_new}\texttt{:N} \ldots \texttt{77, 77}
driver commands:
\texttt{\_\_driver\_box\_rotate}\texttt{:begin} \ldots \texttt{216, 216}
\texttt{\_\_driver\_box\_rotate}\texttt{:end} \ldots \texttt{216, 216}
\texttt{\_\_driver\_box\_scale}\texttt{:begin} \ldots \texttt{216, 216}
\texttt{\_\_driver\_box\_scale}\texttt{:end} \ldots \texttt{216, 216}
\texttt{\_\_driver\_box\_use\_clip}\texttt{:N} \ldots \texttt{215, 215}
\texttt{\_\_driver\_color}\texttt{:ensure}\texttt{:current} \ldots \texttt{216, 216}
e commands:
\texttt{\_c\_e\_fp} \ldots \texttt{187, 190}
eight commands:
\texttt{\_c\_eight} \ldots \texttt{74}
eleven commands:
\texttt{\_c\_eleven} \ldots \texttt{74}
else commands:
\texttt{\_else} \ldots \texttt{24, 37, 75, 75, 75, 75, 90, 143, 143, 143, 143, 179, 179, 196
empty commands:
\c_empty_box \hfill 138, 139
\c_empty_clist \hfill 128
\c_empty_coffin \hfill 147
\c_empty_prop \hfill 135
\c_empty_seq \hfill 119
\c_empty_tl \hfill 105
etex commands:
\etex_\ldots \hfill 9
\etex_\ldots \hfill 196
exp commands:
\exp_after:wN \hfill 33, 33
\exp_args:N \hfill 33
\exp_args:N\{variant\} \hfill 28
\exp_args:Nc \hfill 30, 30
\exp_args:Nccc \hfill 32
\exp_args:Nccx \hfill 32
\exp_args:Nf \hfill 30, 30
\exp_args:Nc \hfill 31, 31
\exp_args:Nnc \hfill 32
\exp_args:NNo \hfill 32, 32, 32
\exp_args:Nn \hfill 32
\exp_args:NNx \hfill 32, 32
\exp_args:Nnnx \hfill 32
\exp_args:Nnx \hfill 32
\exp_args:Nn \hfill 32
\exp_args:Nc \hfill 31
\exp_args:N \hfill 31
\exp_args:Nn \hfill 31
\exp_args:NV \hfill 30, 30
\exp_args:Nv \hfill 30, 30
\exp_args:Nv \hfill 31
\exp_args:N \hfill 31
\exp_args:NXX \hfill 31
\exp_args:Nxx \hfill 31
\_exp_last_two_unbraced:Noo \hfill 33, 33
\_exp_last_unbraced:Nco \hfill 33
\_exp_last_unbraced:Nf \hfill 33
\_exp_last_unbraced:Nnno \hfill 33
\_exp_last_unbraced:NNV \hfill 33
\_exp_last_unbraced:Nn \hfill 33
\_exp_last_unbraced:NNo \hfill 33
\_exp_last_unbraced:Nx \hfill 33
\_exp_last_unbraced:N \hfill 33
\_exp_not:c \hfill 34, 34
\_exp_not:f \hfill 34, 34
\_exp_not:N \hfill 34, 34, 178, 196
\_exp_not:o \hfill 34, 102, 103, 104, 113, 117, 126, 126, 128, 132, 178, 209, 209, 211
\_exp_not:v \hfill 34, 34
\_exp_not:N \hfill 34, 34
\_exp_stop_f: \hfill 35, 35, 35
\_ExplFileVersion \hfill 7
\_ExplFileName \hfill 7
\_ExplFileVersion \hfill 7
\_ExplSyntaxOn \hfill 4, 47, 7, 17, 7, 8
\_ExplSyntaxOff \hfill 4, 4, 7, 7, 7, 8
false \hfill 196
false commands:
\_c_false_bool \hfill 22, 38
fi commands:
\_fi: \hfill 24,
\_if: \hfill 37, 75, 75, 90, 143, 143, 143, 179
fifteen commands:
\_c_fifteen \hfill 74
file commands:
\_file \hfill 173
\_file_add_path:nN \hfill 172, 173, 180
\_g_file_current_name_tl \hfill 173
\_file_if_exist:nTF \hfill 179
\_file_if_exist:nNT \hfill 179
\_file_if_exist:nTF \hfill 173, 173, 173, 175
\_file_if_exist_input:n \hfill 204, 204
\_file_if_exist_input:nTF \hfill 204, 204
\_file_input:n \hfill 173, 173, 173, 204, 204
\_g_file_internal_ior \hfill 179
\_i_file_internal_name_tl \hfill 179
\_file_list \hfill 174, 174
\_file_namesanitize:nn \hfill 179, 179
\_file_pathinclude:n \hfill 173, 174, 174, 204
\_file_pathremove:n \hfill 174, 174
five commands:
\_c_five \hfill 74
floor \hfill 193
foo commands:
\foo:c \hfill 1, 2
\foo:cn \hfill 28
\foo:cV \hfill 28
\foo:N \hfill 1, 2
\foo:Nn \hfill 28
\foo:NV \hfill 28
222
\keys_set:nn \hspace{2em} 168, 168
\keys_set_filter:nn \hspace{2em} 160, 163, 167, 168, 168, 169
\keys_set_filter:nnN \hspace{2em} 170, 170
\keys_set_groups:nn \hspace{2em} 170, 170
\keys_set_known:nn \hspace{2em} 169, 169
\keys_show:nn \hspace{2em} 170, 170
\keys_value_tl \hspace{2em} 168, 168
keyval commands:
  \keyval_parse:NNn \hspace{2em} 172, 172, 172
L
\let \hspace{2em} 1
\ln \hspace{2em} 193
log commands:
  \log_iow \hspace{2em} 179
luatex commands:
  \luatex \hspace{2em} 9
  \luatex_if_engine:TF \hspace{2em} 23, 23
  \luatex_if_engine_p: \hspace{2em} 23
M
mark commands:
  \q_mark \hspace{2em} 26, 26, 46, 105, 105
math commands:
  \c_math_subscript_token \hspace{2em} 54
  \c_math_superscript_token \hspace{2em} 54
  \c_math_toggle_token \hspace{2em} 54
max \hspace{2em} 193
max commands:
  \c_max_dim \hspace{2em} 84, 87
  \c_max_int \hspace{2em} 74
  \c_max_muskip \hspace{2em} 90
  \c_max_register_int \hspace{2em} 74
  \c_max_skip \hspace{2em} 87
meta commands:
  \meta:n \hspace{2em} 164
  \meta:nn \hspace{2em} 164
min \hspace{2em} 193
minus commands:
  \c_minus_inf_fp \hspace{2em} 187, 195
  \c_minus_one \hspace{2em} 74
  \c_minus_zero_fp \hspace{2em} 187
\n \hspace{2em} 196
mode commands:
  \mode_if_horizontal:TF \hspace{2em} 42, 42
  \mode_if_horizontal_p: \hspace{2em} 42, 42
  \mode_if_inner:TF \hspace{2em} 42, 42
msg commands:
  \c__msg_coding_error_text_tl \hspace{2em} 159
  \msg_critical:n \hspace{2em} 152
  \msg_critical:nn \hspace{2em} 152
  \msg_critical:nnn \hspace{2em} 152
  \msg_critical:nnnn \hspace{2em} 152
  \msg_critical:nnnnn \hspace{2em} 152
  \msg_critical:nnnnnn \hspace{2em} 152
  \msg_critical:nnnnnnn \hspace{2em} 152
  \msg_critical:nnnnnnnn \hspace{2em} 152
  \msg_error:n \hspace{2em} 152
  \msg_error:nn \hspace{2em} 152
  \msg_error:nnn \hspace{2em} 152
  \msg_error:nnnn \hspace{2em} 152
  \msg_error:nnnnn \hspace{2em} 152
  \msg_error:nnnnnn \hspace{2em} 152
  \msg_error:nnnnnnn \hspace{2em} 152
  \msg_error:nnnnnnnn \hspace{2em} 152
  \msg_error:nnnnnnnnn \hspace{2em} 152
  \msg_error:nnnnnnnnnn \hspace{2em} 152
  \msg_error:nnnnnnnnnnn \hspace{2em} 152
  \msg_error:nnnnnnnnnnnn \hspace{2em} 152
  \msg_fatal:n \hspace{2em} 151
  \msg_fatal:nn \hspace{2em} 151
  \msg_fatal:nnn \hspace{2em} 151
  \msg_fatal:nnnn \hspace{2em} 151
  \msg_fatal:nnnnn \hspace{2em} 151
  \msg_fatal:nnnnnn \hspace{2em} 151
  \msg_fatal:nnnnnnn \hspace{2em} 151
  \msg_fatal:nnnnnnnn \hspace{2em} 151
  \msg_fatal:nnnnnnnnn \hspace{2em} 151
  \msg_fatal:nnnnnnnnnn \hspace{2em} 151
  \msg_fatal:nnnnnnnnnnn \hspace{2em} 151
  \msg_fatal:nnnnnnnnnnnn \hspace{2em} 151
  \msg_fatal:nnnnnnnnnnnnn \hspace{2em} 151
  \msg_info:n \hspace{2em} 152
  \msg_info:nn \hspace{2em} 152
  \msg_info:nnn \hspace{2em} 152
  \msg_info:nnnn \hspace{2em} 152
  \msg_info:nnnnn \hspace{2em} 152
  \msg_info:nnnnnn \hspace{2em} 152
  \msg_info:nnnnnnn \hspace{2em} 152
  \msg_info:nnnnnnnn \hspace{2em} 152
  \msg_interupt:n \hspace{2em} 153, 155
  \msg_kernel_error:n \hspace{2em} 156
  \msg_kernel_error:nn \hspace{2em} 156
  \msg_kernel_error:nnn \hspace{2em} 156
  \msg_kernel_error:nnnn \hspace{2em} 156
  \msg_kernel_error:nnnnn \hspace{2em} 156
  \msg_kernel_error:nnnnnn \hspace{2em} 156
  \msg_kernel_error:nnnnnnn \hspace{2em} 156
  \msg_kernel_error:nnnnnnnn \hspace{2em} 156
  \msg_kernel_error:nnnnnnnnn \hspace{2em} 156
  \msg_kernel_error:nnnnnnnnnn \hspace{2em} 156
  \msg_kernel_error:nnnnnnnnnnn \hspace{2em} 156
  \msg_kernel_error:nnnnnnnnnnnn \hspace{2em} 156
  \msg_kernel_error:nnnnnnnnnnnnn \hspace{2em} 156
  \msg_kernel_error:nnnnnnnnnnnnn \hspace{2em} 156
  \msg_kernel_error:nnnnnnnnnnnnnn \hspace{2em} 156
  \msg_kernel_error:nnnnnnnnnnnnnn \hspace{2em} 156
  \msg_kernel_error:nnnnnnnnnnnnnnn \hspace{2em} 156
  \msg_kernel_error:nnnnnnnnnnnnnnn \hspace{2em} 156
  \msg_kernel_error:nnnnnnnnnnnnnnn \hspace{2em} 156
  \msg_kernel_error:nnnnnnnnnnnnnnnn \hspace{2em} 156
  \msg_kernel_error:nnnnnnnnnnnnnnnn \hspace{2em} 156
  \msg_keyboard_error:n \hspace{2em} 157
  \msg_keyboard_error:nn \hspace{2em} 157
  \msg_keyboard_error:nnn \hspace{2em} 157
  \msg_keyboard_error:nnnn \hspace{2em} 157
  \msg_keyboard_error:nnnnn \hspace{2em} 157
  \msg_keyboard_error:nnnnnn \hspace{2em} 157
  \msg_keyboard_error:nnnnnnn \hspace{2em} 157
  \msg_keyboard_error:nnnnnnnn \hspace{2em} 157
  \msg_keyboard_error:nnnnnnnnn \hspace{2em} 157
  \msg_keyboard_error:nnnnnnnnnn \hspace{2em} 157
  \msg_keyboard_error:nnnnnnnnnnn \hspace{2em} 157
  \msg_keyboard_error:nnnnnnnnnnnn \hspace{2em} 157
  \msg_keyboard_error:nnnnnnnnnnnnn \hspace{2em} 157
  \msg_keyboard_error:nnnnnnnnnnnnn \hspace{2em} 157
  \msg_keyboard_error:nnnnnnnnnnnnnn \hspace{2em} 157
  \msg_keyboard_error:nnnnnnnnnnnnnnn \hspace{2em} 157
  \msg_keyboard_error:nnnnnnnnnnnnnnn \hspace{2em} 157
  \msg_keyboard_error:nnnnnnnnnnnnnnn \hspace{2em} 157
  \msg_keyboard_error:nnnnnnnnnnnnnnn \hspace{2em} 157
<table>
<thead>
<tr>
<th>Command</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>\skip_if_finite_p:n</td>
<td>86, 86</td>
</tr>
<tr>
<td>\skip_log:N</td>
<td>208, 208</td>
</tr>
<tr>
<td>\skip_log:n</td>
<td>208, 208</td>
</tr>
<tr>
<td>\skip_new:N</td>
<td>85, 85, 85</td>
</tr>
<tr>
<td>.skip_set:c</td>
<td>164</td>
</tr>
<tr>
<td>.skip_set:N</td>
<td>164</td>
</tr>
<tr>
<td>\skip_set:Nn</td>
<td>85, 85</td>
</tr>
<tr>
<td>\skip_set_eq:NN</td>
<td>86, 86</td>
</tr>
<tr>
<td>\skip_show:N</td>
<td>87, 87</td>
</tr>
<tr>
<td>\skip_show:n</td>
<td>87, 87</td>
</tr>
<tr>
<td>\skip_split_finite_else_action:nnNN</td>
<td>208, 208</td>
</tr>
<tr>
<td>\skip_sub:Nn</td>
<td>86, 86</td>
</tr>
<tr>
<td>\skip_use:N</td>
<td>86, 87, 87</td>
</tr>
<tr>
<td>\skip_vertical:N</td>
<td>88, 88, 88</td>
</tr>
<tr>
<td>\skip_vertical:n</td>
<td>88, 88</td>
</tr>
<tr>
<td>\skip_zero:N</td>
<td>85, 85, 88</td>
</tr>
<tr>
<td>\skip_zero_new:N</td>
<td>85, 85</td>
</tr>
<tr>
<td>__str_if_eq_x_return:nn</td>
<td></td>
</tr>
<tr>
<td>__str_if_eq_x:nnTF</td>
<td>109</td>
</tr>
<tr>
<td>__str_if_eq_x:nn</td>
<td>109</td>
</tr>
<tr>
<td>__str_if_eq_x:nnF</td>
<td>107</td>
</tr>
<tr>
<td>__str_if_eq_x:nnTF</td>
<td>107, 107</td>
</tr>
<tr>
<td>__str_if_eq_x:nn</td>
<td>107, 107</td>
</tr>
<tr>
<td>__str_if_eq_x:nnTF</td>
<td>107, 107</td>
</tr>
<tr>
<td>__str_if_eq_x:nn</td>
<td>107, 107</td>
</tr>
<tr>
<td>__str_if_eq_x:nnTF</td>
<td>107, 107</td>
</tr>
<tr>
<td>__str_if_eq_x:nn</td>
<td>107, 107</td>
</tr>
<tr>
<td>__str_if_eq_x:nnTF</td>
<td>107, 107</td>
</tr>
<tr>
<td>__str_if_eq_x:nn</td>
<td>107, 107</td>
</tr>
<tr>
<td>__str_if_eq_x:nn</td>
<td>107, 107</td>
</tr>
<tr>
<td>__str_if_eq_x:nn</td>
<td>107, 107</td>
</tr>
<tr>
<td>__str_if_eq_x:nn</td>
<td>107, 107</td>
</tr>
<tr>
<td>__str_if_eq_x:return:nn</td>
<td>109, 109</td>
</tr>
<tr>
<td>__str_lover_case:n</td>
<td>108, 108, 210</td>
</tr>
<tr>
<td>__str_tail:n</td>
<td>106, 106, 106</td>
</tr>
<tr>
<td>__str_upper_case:n</td>
<td>108, 108, 210</td>
</tr>
<tr>
<td>\text{T}</td>
<td>193</td>
</tr>
<tr>
<td>\tan</td>
<td>194</td>
</tr>
<tr>
<td>\ten commands:</td>
<td></td>
</tr>
<tr>
<td>\c_ten</td>
<td>74</td>
</tr>
<tr>
<td>\c_ten_thousand</td>
<td>74</td>
</tr>
<tr>
<td>\term commands:</td>
<td></td>
</tr>
<tr>
<td>\c_term</td>
<td>174</td>
</tr>
<tr>
<td>\c_term_iow</td>
<td>179</td>
</tr>
<tr>
<td>\c_term_iow</td>
<td>179</td>
</tr>
<tr>
<td>\TeX and \LaTeX commands:</td>
<td></td>
</tr>
<tr>
<td>\box</td>
<td>137</td>
</tr>
<tr>
<td>\copy</td>
<td>137</td>
</tr>
<tr>
<td>\cslname</td>
<td>18</td>
</tr>
<tr>
<td>\dimexpr</td>
<td>91</td>
</tr>
<tr>
<td>\dp</td>
<td>137</td>
</tr>
<tr>
<td>\edef</td>
<td>2</td>
</tr>
<tr>
<td>\endcsname</td>
<td>18</td>
</tr>
<tr>
<td>\endinput</td>
<td>152</td>
</tr>
<tr>
<td>\endtemplate</td>
<td>43</td>
</tr>
<tr>
<td>\errorcontextlines</td>
<td>180</td>
</tr>
<tr>
<td>\escapechar</td>
<td>100, 100, 100</td>
</tr>
<tr>
<td>\expandafter</td>
<td>33</td>
</tr>
<tr>
<td>\halign</td>
<td>43</td>
</tr>
<tr>
<td>\hbox</td>
<td>88</td>
</tr>
<tr>
<td>\hskip</td>
<td>88</td>
</tr>
<tr>
<td>\ht</td>
<td>138</td>
</tr>
<tr>
<td>\ifcase</td>
<td>75</td>
</tr>
<tr>
<td>\ifdim</td>
<td>90</td>
</tr>
<tr>
<td>\ifeof</td>
<td>179</td>
</tr>
<tr>
<td>\ifhbox</td>
<td>143</td>
</tr>
<tr>
<td>\ifnum</td>
<td>75</td>
</tr>
<tr>
<td>\ifodd</td>
<td>75</td>
</tr>
<tr>
<td>\ifvbox</td>
<td>143</td>
</tr>
<tr>
<td>\ifvoid</td>
<td>143</td>
</tr>
<tr>
<td>\ifx</td>
<td>24</td>
</tr>
<tr>
<td>\jobname</td>
<td>105</td>
</tr>
<tr>
<td>\lowercase</td>
<td>95</td>
</tr>
<tr>
<td>\makeatletter</td>
<td>7</td>
</tr>
<tr>
<td>\MakeUppercase</td>
<td>211</td>
</tr>
<tr>
<td>\meaning</td>
<td>17, 55</td>
</tr>
<tr>
<td>\newlinechar</td>
<td>180</td>
</tr>
<tr>
<td>\noexpand</td>
<td>34</td>
</tr>
<tr>
<td>\number</td>
<td>76</td>
</tr>
<tr>
<td>\numexpr</td>
<td>76</td>
</tr>
<tr>
<td>\or</td>
<td>75</td>
</tr>
<tr>
<td>\ProvidesClass</td>
<td>7</td>
</tr>
<tr>
<td>\ProvidesFile</td>
<td>7</td>
</tr>
<tr>
<td>\ProvidesPackage</td>
<td>7</td>
</tr>
<tr>
<td>\read</td>
<td>175</td>
</tr>
</tbody>
</table>

231
\texttt{commands}: 
\texttt{\textbackslash requirepackage} \hfill 176 
\texttt{\textbackslash robustify} \hfill 210 
\texttt{\textbackslash romannumeral} \hfill 75 
\texttt{\textbackslash show} \hfill 17, 104 
\texttt{\textbackslash showtokens} \hfill 105 
\texttt{\textbackslash string} \hfill 55 
\texttt{\textbackslash the} \hfill 66, 83, 87, 89 
\texttt{\textbackslash unexpanded} \hfill 34, 101, 101, 104, 113, 117, 117, 123, 126, 126, 128, 132, 209, 209 
\texttt{\textbackslash unbox} \hfill 141 
\texttt{\textbackslash unhcopy} \hfill 141 
\texttt{\textbackslash unless} \hfill 24 
\texttt{\textbackslash unvbox} \hfill 143 
\texttt{\textbackslash unvcopy} \hfill 143 
\texttt{\textbackslash uppercase} \hfill 96 
\texttt{\textbackslash vbox} \hfill 141 
\texttt{\textbackslash vskip} \hfill 88 
\texttt{\textbackslash vsplit} \hfill 142 
\texttt{\textbackslash vtop} \hfill 141 
\texttt{\textbackslash wd} \hfill 138 
\texttt{\textbackslash write} \hfill 177 

\texttt{\textbackslash tex commands:} 
\texttt{\textbackslash tex} \hfill 9 
\texttt{\textbackslash tex_if:D} \hfill 50 

\texttt{\textbackslash thirteen commands:} 
\texttt{\textbackslash c\textunderscore thirteen} \hfill 74 

\texttt{\textbackslash thirty commands:} 
\texttt{\textbackslash c\textunderscore thirty\textunderscore two} \hfill 74 

\texttt{\textbackslash three commands:} 
\texttt{\textbackslash c\textunderscore three} \hfill 74 

\texttt{\textbackslash tl commands:} 
\texttt{\textbackslash tl\textunderscore (g)clear:N} \hfill 93 
\texttt{\textbackslash tl\textunderscore case:N} \hfill 97 
\texttt{\textbackslash tl\textunderscore case:NTF} \hfill 97, 97 
\texttt{\textbackslash tl\textunderscore case\textunderscore change\textunderscore exclude\textunderscore tl} \hfill 211, 211, 211 
\texttt{\textbackslash tl\textunderscore case\textunderscore change\textunderscore math\textunderscore tl} \hfill 210, 210 
\texttt{\textbackslash tl\textunderscore change\textunderscore case\textunderscore after\textunderscore final\textunderscore _\textunderscore sigma\textunderscore tl} \hfill 212, 212 
\texttt{\textbackslash tl\textunderscore clear:N} \hfill 93, 93 
\texttt{\textbackslash tl\textunderscore clear\textunderscore new:N} \hfill 93, 93 
\texttt{\textbackslash tl\textunderscore concat:NNN} \hfill 93, 93 
\texttt{\textbackslash tl\textunderscore const:NN} \hfill 93, 93 
\texttt{\textbackslash tl\textunderscore count:N} \hfill 97, 100, 101, 101 
\texttt{\textbackslash tl\textunderscore count:n} \hfill 97, 100, 100, 101 
\texttt{\textbackslash tl\textunderscore count\textunderscore tokens:n} \hfill 209, 209 
\texttt{\textbackslash tl\textunderscore expandable\textunderscore lowercase:n} \hfill 209, 209, 209
use commands:
\use:c \hspace{151,18,18,18}
\use:n \hspace{19,19,92,211}
\use:nn \hspace{19,19}
\use:nnn \hspace{19,19}
\use:x \hspace{21,21}
\use_i:nn \hspace{20,20,20}
\use_i:nnn \hspace{20,20,20}
\use_i_delimit_by_q_nil:nw \hspace{21,21}
\use_i_delimit_by_q_recursion_-stop:nu \hspace{21,21}
\use_i_delimit_by_q_stop:nu \hspace{21,21}
\use_i:ii:nn \hspace{20,20,44}
\use_i:ii:nnn \hspace{20,20}
\use_i:iii:nnn \hspace{20,20}
\use:iii:nnn \hspace{20,20}
\use:iv:nnn \hspace{20,20}
\use_none:nn \hspace{21,21,105}
\use_none:nnn \hspace{21,21,21}
\use_none:nnnn \hspace{21,21,21}
\use_none:nnnnn \hspace{21,21,21}
\use_none:nnnnnn \hspace{21,21,21}
\use_none:nnnnnnn \hspace{21,21,21}
\use_none:nnnnnnnn \hspace{21,21,21}
\use_none:nnnnnnnnn \hspace{21,21,21}
\use_none_delimit_by_q-nil:w \hspace{21,21}
\use_none_delimit_by_q_recursion_-stop:w \hspace{21,21}
\use_none_delimit_by_q_stop:w \hspace{21,21}
\_use_none_delimit_by_s_stop:w \hspace{21,21}
\value_required: \hspace{164}
\vbox commands:
\vbox:n \hspace{141,141}
\vbox_gset:Nn \hspace{142}
\vbox_gset:Nw \hspace{142}
\vbox_gset_end: \hspace{142}
\vbox_gset_to_ht:Nnn \hspace{142}
\vbox_gset_top:Nn \hspace{142}
\vbox_set:Nn \hspace{142,142}
\vbox_set:Nw \hspace{142,142}
\vbox_set_end: \hspace{142,142}
\vbox_set_split_to_ht:NNn \hspace{142,142}
\vbox_to_ht:nn \hspace{142,142}
\vbox_to_zero:n \hspace{142,142}
\vbox_top:n \hspace{141,141}
\vbox_unpack:N \hspace{143,143,143}
\vbox_unpack_clear:N \hspace{143}
vcoffin commands:
vcoffin_set:Nnn \hspace{145,145}
vcoffin_set:Nnw \hspace{145,145}
vcoffin_set_end: \hspace{145,145}

void commands:
c_void_box \hspace{136}

xetex commands:
xetex_\hspace{9}
xetex_if_engine:F \hspace{5}
xetex_if_engine:T \hspace{5}
xetex_if_engine:TF \hspace{5,5,23,23}
xetex_if_engine_p: \hspace{23}

zero commands:
c_zero \hspace{74}
c_zero_dim \hspace{84}
c_zero_fp \hspace{187}
c_zero_muskip \hspace{90}
c_zero_skip \hspace{87}