ltlua\texttt{tex}.\texttt{dtx}

(Lua\TeX-specific support)

David Carlisle and Joseph Wright*

2015/10/03

Contents

1 Overview 1

2 Core \TeX functionality 2

3 Plain \TeX interface 2

4 Lua functionality 2
  4.1 Allocators in Lua ........................................... 2
  4.2 Lua access to \TeX{} register numbers .................... 3
  4.3 Module utilities ........................................... 4
  4.4 Callback management ...................................... 4

5 Implementation 5
  5.1 Minimum Lua\TeX{} version ................................. 5
  5.2 Older \LaTeX/Plain \TeX{} setup ............................ 5
  5.3 Attributes .................................................. 7
  5.4 Category code tables ...................................... 7
  5.5 Named Lua functions ...................................... 9
  5.6 Custom whatsits .......................................... 9
  5.7 Lua bytecode registers .................................. 9
  5.8 Lua chunk registers .................................... 9
  5.9 Lua loader .................................................. 10
  5.10 Lua module preliminaries ................................ 11
  5.11 Lua module utilities ................................... 11
  5.12 Accessing register numbers from Lua ..................... 13
  5.13 Attribute allocation ..................................... 14
  5.14 Custom whatsit allocation ................................ 14
  5.15 Bytecode register allocation ............................. 15
  5.16 Lua chunk name allocation ................................ 15
  5.17 Lua callback management .................................. 15

*Significant portions of the code here are adapted/simplified from the packages luatex and\luatexbase written by Heiko Oberdiek, Élie Roux, Manuel Pégourié-Gonnar and Philipp Gesang.
1 Overview

LuaTeX adds a number of engine-specific functions to TeX. Several of these require
set up that is best done in the kernel or need related support functions. This file
provides basic support for LuaTeX at the \LaTeX\ 2ε kernel level plus as a loadable
file which can be used with plain TeX and \LaTeX\.

This file contains code for both TeX (to be stored as part of the format) and
Lua (to be loaded at the start of each job). In the Lua code, the kernel uses the
namespace \texttt{luatexbase}.

The following \texttt{\count} registers are used here for register allocation:

- \texttt{\e@alloc@attribute@count} Attributes (default 258)
- \texttt{\e@alloc@ccodetable@count} Category code tables (default 259)
- \texttt{\e@alloc@luafunction@count} Lua functions (default 260)
- \texttt{\e@alloc@whatsit@count} User whatsits (default 261)
- \texttt{\e@alloc@bytecode@count} Lua bytecodes (default 262)
- \texttt{\e@alloc@luachunk@count} Lua chunks (default 263)

(\texttt{\count 256} is used for \texttt{\newmarks} allocation and \texttt{\count 257} is used for
\texttt{\newXeTeXintercharclass} with XeTeX, with code defined in \texttt{ltfinal.dtx}).
With any \LaTeX\ 2ε kernel from 2015 onward these registers are part of the block
in the extended area reserved by the kernel (prior to 2015 the \LaTeX\ 2ε kernel did
not provide any functionality for the extended allocation area).

2 Core TeX functionality

The commands defined here are defined for possible inclusion in a future \LaTeX\ format, however also extracted to the file \texttt{ltlatex.tex} which may be used with
older \LaTeX\ formats, and with plain TeX.

- \texttt{\newattribute{⟨attribute⟩}}
  Defines a named \texttt{attribute}, indexed from 1 (i.e. \texttt{\attribute0} is never defined).
  Attributes initially have the marker value “\texttt{\textasciitilde}FFFFF” (‘unset’) set by the engine.
- \texttt{\newcatcodetable{⟨catcodetable⟩}}
  Defines a named \texttt{catcodetable}, indexed from 1 (\texttt{\catcodetable0} is never as-
  signed). A new catcode table will be populated with exactly those values assigned
  by IniTeX (as described in the LuaTeX manual).
- \texttt{\newluafunction{⟨function⟩}}
  Defines a named \texttt{luafunction}. indexed from 1. (Lua indexes tables from 1 so
  \texttt{\luafunction0} is not available).
- \texttt{\newwhatsit{⟨whatsit⟩}}
  Defines a custom \texttt{whatsit}, indexed from 1.
- \texttt{\newluabytecode{⟨bytecode⟩}}
  Allocates a number for lua bytecode register, indexed from 1.
- \texttt{\newluachunkname{⟨chunkname⟩}}
  Allocates a number for lua chunk register, indexed from 1. Also enters the name of the
  register (without backslash) into the \texttt{lua.name} table to be used in stack traces.
- \texttt{\catcodetable@initex}
- \texttt{\catcodetable@string}
- \texttt{\catcodetable@latex}
- \texttt{\catcodetable@atletter}
latex and atletter tables set the full Unicode range to the codes predefined by the kernel.

\setattribute \setattribute{\langle attribute\rangle}\{\langle value\rangle\}
\unsetattribute{\langle attribute\rangle}

Set and unset attributes in a manner analogous to \setlength. Note that attributes take a marker value when unset so this operation is distinct from setting the value to zero.

3 Plain \TeX interface

The \ltluatex interface may be used with plain \TeX using \texttt{\input{ltluatex}} this inputs \texttt{ltluatex.tex} which inputs \texttt{etex.src} (or \texttt{etex.sty} if used with \texttt{\LaTeX}) if it is not already input, and then defines some internal commands to allow the \ltluatex interface to be defined.

The \luatexbase package interface may also be used in plain \TeX, as before, by inputting the package \texttt{\input luatexbase.sty}. The new version of \luatexbase is based on this \ltluatex code but implements a compatibility layer providing the interface of the original package.

4 Lua functionality

4.1 Allocators in Lua

\begin{itemize}
  \item \texttt{\newattribute} \texttt{\luatexbase.new_attribute(\langle attribute\rangle)}
  \texttt{\new_attribute} Returns an allocation number for the \langle attribute\rangle, indexed from 1. The attribute will be initialised with the marker value \texttt{\textasciitilde7FFFFFFF (\textasciitildeunset\texttt)}. The attribute allocation sequence is shared with the \TeX code but this function does not define a token using \texttt{\attribute\texttt{def}}. The attribute name is recorded in the \texttt{attributes table}. A metatable is provided so that the table syntax can be used consistently for attributes declared in \TeX or lua.
  \item \texttt{\new_whatsit} \texttt{\luatexbase.new_whatsit(\langle whatsit\rangle)}
  \texttt{\new_whatsit} Returns an allocation number for the custom \langle whatsit\rangle, indexed from 1.
  \item \texttt{\new_bytecode} \texttt{\luatexbase.new_bytecode(\langle bytecode\rangle)}
  \texttt{\new_bytecode} Returns an allocation number for a bytecode register, indexed from 1. The optional \langle name\rangle argument is just used for logging.
  \item \texttt{\new_chunkname} \texttt{\luatexbase.new_chunkname(\langle chunkname\rangle)}
  \texttt{\new_chunkname} Returns an allocation number for a lua chunk name for use with \texttt{\directlua} and \texttt{\latexlua}, indexed from 1. The number is returned and also \langle name\rangle argument is added to the lua.name array at that index.
\end{itemize}

4.2 Lua access to \TeX register numbers

\begin{itemize}
  \item \texttt{\registernumber} \texttt{\luatexbase.registernumber(\langle name\rangle)}
  \texttt{\registernumber} Sometimes (notably in the case of Lua attributes) it is necessary to access a register by number that has been allocated by \TeX. This package provides a function to look up the relevant number using Lua\TeX\xspace’s internal tables. After for example \texttt{\newattribute\myattrib, \myattrib} would be defined by (say) \texttt{\myattrib=\attribute15. \luatexbase.registernumber("\myattrib")} would then return the register number, 15 in this case. If the string passed as
argument does not correspond to a token defined by \attributedef, \countdef
or similar commands, the Lua value false is returned.

As an example, consider the input:

\newcommand\test[1]{%
% \typeout{#1: \expandafter\meaning\csname#1\endcsname^^J
% \space\space\space\space
% \directlua{tex.write(luatexbase.registernumber("#1") or "bad input")}%
% }
\test{undefinedrubbish}
\test{space}
\test{hbox}
\test{@MM}
\test{@tempdima}
\test{@tempdimb}
\test{strutbox}
\test{sixt@@n}
\attributedef\myattr=12
\myattr=200
\test{myattr}

If the demonstration code is processed with \LaTeX then the following would
be produced in the log and terminal output.

undefinedrubbish: \relax
  bad input
space: macro:->
  bad input
hbox: \hbox
  bad input
@MM: \mathchar"4E20
  20000
@tempdima: \dimen14
  14
@tempdimb: \dimen15
  15
strutbox: \char"B
  11
sixt@@n: \char"10
  16
myattr: \attribute12
  12

Notice how undefined commands, or commands unrelated to registers do not
produce an error, just return false and so print bad input here. Note also that
commands defined by \newbox work and return the number of the box register
even though the actual command holding this number is a \chardef defined token (there is no \boxdef).

4.3 Module utilities

provide_module luatexbase.provides_module(⟨info⟩)
This function is used by modules to identify themselves; the info should be a table containing information about the module. The required field name must contain the name of the module. It is recommended to provide a field date in the usual \TeX format yyyy/mm/dd. Optional fields version (a string) and description may be used if present. This information will be recorded in the log. Other fields are ignored.

module_info luatexbase.module_info(⟨module⟩,⟨text⟩)
module_warning luatexbase.module_warning(⟨module⟩,⟨text⟩)
module_error luatexbase.module_error(⟨module⟩,⟨text⟩)
These functions are similar to \TeX's \PackageError, \PackageWarning and \PackageInfo in the way they format the output. No automatic line breaking is done, you may still use \n as usual for that, and the name of the package will be prepended to each output line.

Note that luatexbase.module_error raises an actual Lua error with error(), which currently means a call stack will be dumped. While this may not look pretty, at least it provides useful information for tracking the error down.

4.4 Callback management

add_to_callback luatexbase.add_to_callback(⟨callback⟩,⟨function⟩,⟨description⟩) Registers the ⟨function⟩ into the ⟨callback⟩ with a textual ⟨description⟩ of the function. Functions are inserted into the callback in the order loaded.

remove_from_callback luatexbase.remove_from_callback(⟨callback⟩,⟨description⟩) Removes the callback function with ⟨description⟩ from the ⟨callback⟩. The removed function and its description are returned as the results of this function.

in_callback luatexbase.in_callback(⟨callback⟩,⟨description⟩) Checks if the ⟨description⟩ matches one of the functions added to the list for the ⟨callback⟩, returning a boolean value.

disable_callback luatexbase.disable_callback(⟨callback⟩) Sets the ⟨callback⟩ to false as described in the Lua\TeX manual for the underlying callback.register built-in. Callbacks will only be set to false (and thus be skipped entirely) if there are no functions registered using the callback.

callback_descriptions A list of the descriptions of functions registered to the specified callback is returned. {} is returned if there are no functions registered.

create_callback luatexbase.create_callback(⟨name⟩,metatype,⟨default⟩) Defines a user defined callback. The last argument is a default function of false.

call_callback luatexbase.call_callback(⟨name⟩,...) Calls a user defined callback with the supplied arguments.

5 Implementation

1 \texttt{	extasciitilde 2ekernel | tex | latexrelease}
2 \texttt{(2ekernel | latexrelease)\ifx\directlua\@undefined\else\@undefined\else
5.1 Minimum Lua\TeX\ version

Lua\TeX\ has changed a lot over time. In the kernel support for ancient versions is not provided: trying to build a format with a very old binary therefore gives some information in the log and loading stops. The cut-off selected here relates to the tree-searching behaviour of require(): from version 0.60, Lua\TeX\ will correctly find Lua files in the texmf tree without 'help'.

\begin{verbatim}
\ifnum\luatexversion<60 \%
  \wlog{***************************************************}
  \wlog{* Lua\TeX\ version too old for ltluatex support *}
  \wlog{***************************************************}
  \expandafter\endinput
\fi
\end{verbatim}

5.2 Older \La\TeX/Plain \TeX\ setup

Older \La\TeX\ formats don’t have the primitives with ‘native’ names: sort that out. If they already exist this will still be safe.

\begin{verbatim}
\directlua{tex.enableprimitives('',tex.extraprimitives('luatex'))}
\if\e@alloc\@undefined
  In pre-2014 \La\TeX, or plain \TeX, load etex.{sty,src}.
  \if\documentclass\@undefined
    \if\loccount\@undefined
      \input{etex.src}%
    \fi
    \catcode'@=11 \%
    \outer\expandafter\def\csname newfam\endcsname
    {\alloc@8\fam\chardef\et@xmaxfam}
  \else
    \RequirePackage{etex}
    \expandafter\def\csname newfam\endcsname
    {\alloc@8\fam\chardef\et@xmaxfam}
    \expandafter\let\expandafter\new@mathgroup\csname newfam\endcsname
  \fi
  \endinput
\else
  \RequirePackage{etex}
  \expandafter\def\csname newfam\endcsname
  {\alloc@8\fam\chardef\et@xmaxfam}
  \expandafter\let\expandafter\new@mathgroup\csname newfam\endcsname
  \endinput
\fi
\end{verbatim}

5.2.1 Fixes to etex.src/etex.sty

These could and probably should be made directly in an update to etex.src which already has some luatex-specific code, but does not define the correct range for luatex.

\begin{verbatim}
% 2015-07-13 higher range in luatex
\edef \et@xmaxregs {\if\directlua\@undefined 32768\else 65536\fi}
% luatex/xetex also allow more math fam
\edef \et@xmaxfam {\if\Umathchar\@undefined\sixt@@n\else\@cclvi\fi}
\count270=\et@xmaxregs \% locally allocates \count registers
\count271=\et@xmaxfam \% ditto for \dimen registers
\count272=\et@xmaxfams \% ditto for \skip registers
\count273=\et@xmaxfams \% ditto for \muskip registers
\count274=\et@xmaxfams \% ditto for \box registers
\end{verbatim}
\count 275=\et@xmaxregs % ditto for \toks registers
\count 276=\et@xmaxregs % ditto for \marks classes

and 256 or 16 fam. (Done above due to plain/\eTeX{} differences in \ltlatex{}.)

End of proposed changes to \etex{}\src

5.2.2 luatex specific settings

Switch to global cf \luatex\sty to leave room for inserts not really needed for
\latex{} but possibly most compatible with existing use.

\expandafter\let\csname newcount\expandafter\expandafter\endcsname
\csname globcount\endcsname
\expandafter\let\csname newdimen\expandafter\expandafter\endcsname
\csname globdimen\endcsname
\expandafter\let\csname newskip\expandafter\expandafter\endcsname
\csname globskip\endcsname
\expandafter\let\csname newbox\expandafter\expandafter\endcsname
\csname globbox\endcsname

Define \e@alloc as in \latex{} (the existing macros in \etex\src hard to extend
to further register types as they assume specific 26x and 27x count range. For
compatibility the existing register allocation is not changed.

\chardef\e@alloc@top=65535
\let\e@alloc@chardef\chardef
\def\e@alloc#1#2#3#4#5#6{%  
  \global\advance#3\@ne
  \e@ch@ck{#3}{#4}{#5}#1%
  \allocationnumber#3\relax
  \global#2#6\allocationnumber  
  \wlog{\string#6=\string#1\the\allocationnumber}%
\gdef\e@ch@ck#1#2#3#4{%  
  \ifnum#1<#2\else
    \ifnum#1=#2\relax
      #1\@cclvi
      \ifx\count#4\advance#1 10 \fi
    \fi
    \ifnum#1<#3\relax
      \else
        \errmessage{No room for a new \string#4}%
    \fi
  \fi
\}

Two simple \eTeX{} macros used in \ltlatex\sty.
\long\def\@gobble#1{}
\long\def\@firstofone#1{#1}

\% Fix up allocations not to clash with \etex\src/.
End of conditional setup for plain \TeX / old \LaTeX.

5.3 Attributes

\newattribute

As is generally the case for the \LaTeX registers we start here from 1. Notably, some code assumes that \texttt{\attribute0} is never used so this is important in this case.

\fi
\langle /tex \rangle

\countdef \e@alloc@attribute@count = 258
\ifx \e@alloc@attribute@count \undefined
\def \newattribute #1 %
\\e@alloc \attribute \attributedef
\\e@alloc@attribute@count \m@ne \e@alloc@top #1 %
\}
\\e@alloc@attribute@count = \z@ %
\else
\def \newattribute #1 %
\\e@alloc@attribute@count \m@ne \e@alloc\attribute\attributedef
\\e@alloc@attribute@count \m@ne \e@alloc@top #1 %
\}
\\e@alloc@attribute@count = \z@ %
\fi

\setattribute
\unsetattribute
Handy utilities.

\def \setattribute #1 #2 { #1 = \numexpr #2 \relax }
\def \unsetattribute #1 { #1 = \numexpr -7FFFFFFF \relax }

5.4 Category code tables

\newcatcodetable

Category code tables are allocated with a limit half of that used by \LaTeX for everything else. At the end of allocation there needs to be an initialisation step. Table 0 is already taken (it's the global one for current use) so the allocation starts at 1.

\countdef \e@alloc@ccodetable@count = 259
\ifx \e@alloc@ccodetable@count \undefined
\def \newcatcodetable #1 %
\\e@alloc \catcodetable \chardef
\\e@alloc@ccodetable@count \m@ne \char\langle 8000 \rangle #1 %
\\initcatcodetable \allocationnumber
\}
\\e@alloc@ccodetable@count = \z@ %
\else
\def \newcatcodetable #1 %
\\e@alloc \catcodetable \chardef
\\e@alloc@ccodetable@count \m@ne \char\langle 8000 \rangle #1 %
\\initcatcodetable \allocationnumber
\}
\\e@alloc@ccodetable@count = \z@ %
\fi

\catcodetable@initex
\catcodetable@string
\catcodetable@latex
\catcodetable@atletter
Save a small set of standard tables. The Unicode data is read here in a group avoiding any global definitions: that needs a bit of effort so that in package/plain mode there is no effect on any settings already in force.

\newcatcodetable \catcodetable@initex
\newcatcodetable \catcodetable@string
\begingroup
\def \setrangecatcode #1 #2 #3 { %
\ifnum #1 > #2 %
\expandafter \@gobble
\else
\expandafter \@firstofone
\fi %
\setrangecatcode #1 #2 #3 %
\endgroup

\catcode\#1 = \z@ %
5.5 Named Lua functions

\newluafunction Much the same story for allocating Lua\TeX{} functions except here they are just numbers so are allocated in the same way as boxes. Lua index from 1 so once again slot 0 is skipped.

\newwhatsit These are only settable from Lua but for consistency are definable here.
5.7 Lua bytecode registers

These are only settable from Lua but for consistency are definable here.

5.8 Lua chunk registers

As for bytecode registers, but in addition we need to add a string to the \texttt{lua.name} table to use in stack tracing. We use the name of the command passed to the allocator, with no backslash.

5.9 Lua loader

Load the Lua code at the start of every job. For the conversion of \TeX into numbers at the Lua side we need some known registers; for convenience we use a set of systematic names, which means using a group around the Lua loader.
\directlua{require("ltluatex")}
\endgroup
\langle 2ekernel\rangle
\EndIncludeInRelease
\%
\changes{v1.0b}{2015/10/02}{Fix backing out of \TeX{} code}
\%
\changes{v1.0c}{2015/10/02}{Allow backing out of Lua code}
\langle latexrelease \rangle
\IncludeInRelease{0000/00/00}
\langle latexrelease \rangle
\{\newluafunction\} \texttt{LuaTeX}%
\langle latexrelease \rangle
\let\e@alloc@attribute@count\@undefined
\langle latexrelease \rangle
\let\newattribute\@undefined
\langle latexrelease \rangle
\let\setattribute\@undefined
\langle latexrelease \rangle
\let\unsetattribute\@undefined
\langle latexrelease \rangle
\let\e@alloc@ccodetable@count\@undefined
\langle latexrelease \rangle
\let\newcatcodetable\@undefined
\langle latexrelease \rangle
\let\catcodetable@initex\@undefined
\langle latexrelease \rangle
\let\catcodetable@string\@undefined
\langle latexrelease \rangle
\let\catcodetable@latex\@undefined
\langle latexrelease \rangle
\let\catcodetable@atletter\@undefined
\langle latexrelease \rangle
\let\e@alloc@luafunction@count\@undefined
\langle latexrelease \rangle
\let\newluafunction\@undefined
\langle latexrelease \rangle
\let\e@alloc@whatsit@count\@undefined
\langle latexrelease \rangle
\let\newwhatsit\@undefined
\langle latexrelease \rangle
\let\e@alloc@bytecode@count\@undefined
\langle latexrelease \rangle
\let\newluabytecode\@undefined
\langle latexrelease \rangle
\let\e@alloc@luachunk@count\@undefined
\langle latexrelease \rangle
\directlua{luatexbase.uninstall()}
\langle latexrelease \rangle
\EndIncludeInRelease
\langle 2ekernel \rangle latexrelease \fi
\langle /2ekernel \rangle tex latexrelease

5.10 Lua module preliminaries

\langle \texttt{lua} \rangle

Some set up for the Lua module which is needed for all of the Lua functionality added here.

\texttt{luatexbase} Set up the table for the returned functions. This is used to expose all of the public functions.

\texttt{luatexbase} = \texttt{luatexbase} or { }

local \texttt{luatexbase} = \texttt{luatexbase}

Some Lua best practice: use local versions of functions where possible.

local \texttt{string.gsub} = \texttt{string.gsub}
local \texttt{tex.count} = \texttt{tex.count}
local \texttt{tex.setattribute} = \texttt{tex.setattribute}
local \texttt{tex.setcount} = \texttt{tex.setcount}
local \texttt{texio.write_nl} = \texttt{texio.write_nl}

11
5.11 Lua module utilities

5.11.1 Module tracking

modules To allow tracking of module usage, a structure is provided to store information and to return it.

local modules = modules or {

provides_module Local function to write to the log.

local function luatexbase_log(text)
  texio_write_nl("log", text)
end

\% \begin{macrocode}
local function provides_module(info)
  if not (info and info.name) then
    luatexbase_error("Missing module name for provides_modules")
    return
  end
local function spaced(text)
  return text and (" " .. text) or ""
end
luatexbase_log("Lua module: " .. info.name
  .. spaced(info.date)
  .. spaced(info.version)
  .. spaced(info.description)
}
modules[info.name] = info
end
luatexbase.provides_module = provides_module

5.11.2 Module messages

There are various warnings and errors that need to be given. For warnings we can get exactly the same formatting as from \TeX. For errors we have to make some changes. Here we give the text of the error in the \LaTeX format then force an error from Lua to halt the run. Splitting the message text is done using \textbackslash n which takes the place of \MessageBreak.

First an auxiliary for the formatting: this measures up the message leader so we always get the correct indent.

local function msg_format(mod, msg_type, text)
  local leader = ""
  local cont
  if mod == "LaTeX" then
    cont = string_gsub(leader, ".", " ")
    leader = leader .. "LaTeX: "
  else
    first_head = leader .. "Module " .. msg_type
    cont = "(" .. mod .. ")"

12
.. string_gsub(first_head, ".", " ")

first_head = leader .. "Module " .. mod .. " " .. msg_type .. ":" 
end

if msg_type == "Error" then
  first_head = "\n" .. first_head
end

if string.sub(text,-1) ~= "\n" then
  text = text .. " 
end

return first_head .. " 
.. string_gsub( 
  text
  .. "on input line 
  .. tex.inputlineno, "\n", "\n" .. cont .. " 
  )
  .. "\n"
end

module_info module_warning module_error

local function module_info(mod, text)
  texio_write_nl("log", msg_format(mod, "Info", text))
end

luatexbase.module_info = module_info

local function module_warning(mod, text)
  texio_write_nl("term and log",msg_format(mod, "Warning", text))
end

luatexbase.module_warning = module_warning

local function module_error(mod, text)
  error(msg_format(mod, "Error", text))
end

luatexbase.module_error = module_error

Dedicated versions for the rest of the code here.

local function luatexbase_warning(text)
  module_warning("luatexbase", text)
end

local function luatexbase_error(text)
  module_error("luatexbase", text)
end

5.12 Accessing register numbers from Lua

Collect up the data from the \TeX{} level into a Lua table: from version 0.80, Lua\TeX{} makes that easy.

local luaregisterbasetable = { }
local registermap = {
  attributezero = "assign_attr" ,
  charzero = "char_given" ,
  CountZero = "assign_int" ,
  dimenzero = "assign_dimen" ,
  mathcharzero = "math_given" ,
  muskipzero = "assign_mu_skip" ,
  skipzero = "assign_skip" ,
local i, j
local createtoken
if tex.luatexversion >79 then
  createtoken = newtoken.create
end
local hashtokens = tex.hashtokens
local luatexversion = tex.luatexversion
for i,j in pairs (registermap) do
  if luatexversion < 80 then
    luaregisterbasetable[hashtokens()[i][1]] =
    hashtokens()[i][2]
  else
    luaregisterbasetable[j] = createtoken(i).mode
  end
end
local registernumber
if luatexversion < 80 then
  function registernumber(name)
    local nt = hashtokens()[name]
    if(nt and luaregisterbasetable[nt[1]]) then
      return nt[2] - luaregisterbasetable[nt[1]]
    else
      return false
    end
  end
else
  function registernumber(name)
    local nt = createtoken(name)
    if(luaregisterbasetable[nt.cmdname]) then
      return nt.mode - luaregisterbasetable[nt.cmdname]
    else
      return false
    end
  end
end
luatexbase.registernumber = registernumber

5.13 Attribute allocation

As attributes are used for Lua manipulations its useful to be able to assign from this end.

local attributes=setmetatable(
  {},
  {__index = function(t,key)
    return registernumber(key) or nil
  },
local function new_attribute(name)
    tex_setcount("global", "e@alloc@attribute@count",
        tex_count["e@alloc@attribute@count"] + 1)
    if tex_count["e@alloc@attribute@count"] > 65534 then
        luatexbase_error("No room for a new \attribute")
        return -1
    end
    attributes[name] = tex_count["e@alloc@attribute@count"]
    luatexbase_log("Lua-only attribute " .. name .. " = " ..
        tex_count["e@alloc@attribute@count"])
    return tex_count["e@alloc@attribute@count"]
end
luatexbase.new_attribute = new_attribute

5.14 Custom whatsit allocation

local function new_whatsit(name)
    tex_setcount("global", "e@alloc@whatsit@count",
        tex_count["e@alloc@whatsit@count"] + 1)
    if tex_count["e@alloc@whatsit@count"] > 65534 then
        luatexbase_error("No room for a new custom whatsit")
        return -1
    end
    luatexbase_log("Custom whatsit " .. (name or ") .. " = " ..
        tex_count["e@alloc@whatsit@count"])
    return tex_count["e@alloc@whatsit@count"]
end
luatexbase.new_whatsit = new_whatsit

5.15 Bytecode register allocation

local function new_bytecode(name)
    tex_setcount("global", "e@alloc@bytecode@count",
        tex_count["e@alloc@bytecode@count"] + 1)
    if tex_count["e@alloc@bytecode@count"] > 65534 then
        luatexbase_error("No room for a new bytecode register")
        return -1
    end
    luatexbase_log("Lua bytecode " .. (name or ") .. " = " ..
        tex_count["e@alloc@bytecode@count"])
    return tex_count["e@alloc@bytecode@count"]
end
luatexbase.new_bytecode = new_bytecode
5.16 Lua chunk name allocation

new_chunkname

As for bytecode registers but also store the name in the lua.name table.

```lua
local function new_chunkname(name)
    tex_setcount("global", "e@alloc@luachunk@count",
        tex_count["e@alloc@luachunk@count"] + 1)
    local chunkname_count = tex_count["e@alloc@luachunk@count"]
    chunkname_count = chunkname_count + 1
    if chunkname_count > 65534 then
        luatexbase_error("No room for a new chunkname")
        return -1
    end
    lua.name[chunkname_count]=name
    luatexbase_log("Lua chunkname " .. (name or ") .. " = " ..
        chunkname_count .. "\n")
    return chunkname_count
end
luatexbase.new_chunkname = new_chunkname
```

5.17 Lua callback management

The native mechanism for callbacks in Lua allows only one per function. That is extremely restrictive and so a mechanism is needed to add and remove callbacks from the appropriate hooks.

5.17.1 Housekeeping

The main table: keys are callback names, and values are the associated lists of functions. More precisely, the entries in the list are tables holding the actual function as func and the identifying description as description. Only callbacks with a non-empty list of functions have an entry in this list.

```lua
local callbacklist = callbacklist or { }
local list, data, exclusive, simple = 1, 2, 3, 4
local types = {
    list = list,
    data = data,
    exclusive = exclusive,
    simple = simple,
}

Numerical codes for callback types, and name-to-value association (the table keys are strings, the values are numbers).
```

Now, list all predefined callbacks with their current type, based on the LuaTeX manual version 0.80. A full list of the currently-available callbacks can be obtained using

```latex
\directlua{
    for i,... in pairs(callback.list()) do
        texio.write_nl("- " .. i)
    end
}
\bye
```
in plain LuaTeX. (Some undocumented callbacks are omitted as they are to be
removed.)

```lua
local callbacktypes = callbacktypes or {

Section 4.1.1: file discovery callbacks.

find_read_file = exclusive,
find_write_file = exclusive,
find_font_file = data,
find_output_file = data,
find_format_file = data,
find_vf_file = data,
find_map_file = data,
find_enc_file = data,
find_sfd_file = data,
find_pk_file = data,
find_data_file = data,
find_opentype_file = data,
find_truetype_file = data,
find_type1_file = data,
find_image_file = data,

Section 4.1.2: file reading callbacks.

open_read_file = exclusive,
read_font_file = exclusive,
read_vf_file = exclusive,
read_map_file = exclusive,
read_enc_file = exclusive,
read_sfd_file = exclusive,
read_pk_file = exclusive,
read_data_file = exclusive,
read_opentype_file = exclusive,
read_truetype_file = exclusive,
read_type1_file = exclusive,

Section 4.1.3: data processing callbacks.

process_input_buffer = data,
process_output_buffer = data,
process_jobname = data,
token_filter = exclusive,

Section 4.1.4: node list processing callbacks.

buildpage_filter = simple,
pref_linebreak_filter = list,
linebreak_filter = list,
post_linebreak_filter = list,
hpacak_filter = list,
vpack_filter = list,
pref_output_filter = list,
hyphenate = simple,
ligaturing = simple,
kerning = simple,
mlist_to_hlist = list,

Section 4.1.5: information reporting callbacks.

pre_dump = simple,
start_run = simple,
```
Section 4.1.6: PDF-related callbacks.

```
finish_pdffile = data,
finish_pdfpage = data,
```

Section 4.1.7: font-related callbacks.

```
define_font = exclusive,
```

Undocumented callbacks which are likely to get documented.

```
find_cidmap_file = data,
pdf_stream_filter_callback = data,
```
Handler for list callbacks.

```lua
local function list_handler(name)
  return function(head, ...)  
  local ret
  local alltrue = true
  local i
  for _,i in ipairs(callbacklist[name]) do
    ret = i.func(head, ...)
    if ret == false then
      luatexbase_warning("Function ‘i.description’ returned false\n"
       .. "in callback ‘name’"
      )
      break
    end
    alltrue = false
    head = ret
  end
  return alltrue and true or head
end
```

Handler for simple callbacks.

```lua
local function simple_handler(name)
  return function(...)  
  local i
  for _,i in ipairs(callbacklist[name]) do
    i.func(...)
  end
end
```

Keep a handlers table for indexed access.

```lua
local handlers = {
  [data] = data_handler,
  [exclusive] = exclusive_handler,
  [list] = list_handler,
  [simple] = simple_handler,
}
```

### 5.17.3 Public functions for callback management

Defining user callbacks perhaps should be in package code, but impacts on `add_to_callback`. If a default function is not required, may may be declared as `false`. First we need a list of user callbacks.

```lua
local user_callbacks_defaults = { }
```

create_callback  The allocator itself.

```lua
local function create_callback(name, ctype, default)
  if not name or
end
```
name == "" or
callbacktypes[name] or
not(default == false or type(default) == "function")
then
  luatexbase_error("Unable to create callback " .. name)
end
user_callbacks_defaults[name] = default
callbacktypes[name] = types[ctype]
end
luatexbase.create_callback = create_callback

call_callback  Call a user defined callback. First check arguments.
local function call_callback(name,...)
  if not name or
    name == "" or
    user_callbacks_defaults[name] == nil
  then
    luatexbase_error("Unable to call callback " .. name)
  end
  local l = callbacklist[name]
  local f
  if not l then
    f = user_callbacks_defaults[name]
    if l == false then
      return nil
    end
  else
    f = handlers[callbacktypes[name]](name)
  end
  return f(...)
end
luatexbase.call_callback=call_callback

add_to_callback  Add a function to a callback. First check arguments.
local function add_to_callback(name, func, description)
  if
    not name or
    name == "" or
    not callbacktypes[name] or
    type(func) ~= "function" or
    not description or
    description == ""
  then
    luatexbase_error("Unable to register callback.\n\n" .. "Correct usage:\n" .. "add_to_callback(<callback>, <function>, <description>)"
  )
  return
end

Then test if this callback is already in use. If not, initialise its list and register the proper handler.
local l = callbacklist[name]
if l == nil then
l = { }
callbacklist[name] = l

If it is not a user defined callback use the primitive callback register.

if user_callbacks_defaults[name] == nil then
callback_register(name, handlers[callbacktypes[name]](name))
end

Actually register the function and give an error if more than one exclusive one is registered.

local f = {
  func = func,
  description = description,
}
local priority = #l + 1
if callbacktypes[name] == exclusive then
  if #l == 1 then
    luatexbase_error("Cannot add second callback to exclusive function\n" .. name .. ")
  end
end

local l = callbacklist[name]
if not l then
  luatexbase_error("No callback list for " .. name .. ")
end

Loop over the callback's function list until we find a matching entry. Remove it and check if the list is empty: if so, unregister the callback handler.

remove_from_callback Remove a function from a callback. First check arguments.

local function remove_from_callback(name, description)
  if not name or
    name == "" or
  not callbacktypes[name] or
    not description or
  description == "" then
    luatexbase_error("Unable to remove function from callback.\n" .. "Correct usage:\n" .. "remove_from_callback(<callback>, <description>)")
  return
end

local l = callbacklist[name]
if not l then
  luatexbase_error("No callback list for " .. name .. ")
end

Loop over the callback's function list until we find a matching entry. Remove it and check if the list is empty: if so, unregister the callback handler.
local index = false
local i,j
local cb = {}
for i,j in ipairs(l) do
    if j.description == description then
        index = i
        break
    end
end
if not index then
    luatexbase_error(  
        "No callback \" .. description .. ", registered for \" ..  
        name .. \\
        \"\n")
    return
end
cb = l[index]
table.remove(l, index)
luatexbase_log(  
    "Removing \" .. description .. \" from \" .. name .. \"."
)  
if #l == 0 then
    callbacklist[name] = nil
    callback_register(name, nil)
end
return cb.func,cb.description
end
luatexbase.remove_from_callback = remove_from_callback

in_callback Look for a function description in a callback.
local function in_callback(name, description)
    if not name  
        or name == ""  
        or not callbacktypes[name]  
        or not description then
        return false
    end
    local i
    for _, i in pairs(callbacklist[name]) do
        if i.description == description then
            return true
        end
    end
    return false
end
luatexbase.in_callback = in_callback

disable_callback As we subvert the engine interface we need to provide a way to access this func-
local function disable_callback(name)
if(callbacklist[name] == nil) then
    callback_register(name, false)
else
    luatexbase_error("Callback list for " .. name .. " not empty")
end

callback_descriptions  List the descriptions of functions registered for the given callback.

```lua
local function callback_descriptions (name)
    local d = {}
    if not name
        or name == ""
        or not callbacktypes[name]
        then
        return d
    else
    local i
    for k, i in pairs(callbacklist[name] or {}) do
        d[k] = i.description
    end
    end
    return d
end
luatexbase.callback_descriptions = callback_descriptions
```

uninstall  Unlike at the \TeX{} level, we have to provide a back-out mechanism here at the same time as the rest of the code. This is not meant for use by anything other than \texttt{latexrelease}: as such this is \textit{deliberately} not documented for users!

```lua
local function uninstall()
    module_info(
        "luatexbase",
        "Uninstalling kernel luatexbase code"
    )
    callback.register = callback_register
    luatexbase = nil
    end
luatexbase.uninstall = uninstall
```

\texttt{/lua}

Reset the catcode of \texttt{\@}.

\texttt{(tex)\textbackslash catcode\texttt{\@}=\textbackslash etatcatcode\relax}