luatexbase.dtx

(Lua\TeX\-specific support, luatexbase interface)

David Carlisle and Joseph Wright*

2015/10/04

Contents

1 Overview 1

2 The \texttt{luatexbase} package interface 1
  2.1 Catcode tables ................................................. 1
  2.2 Lua Callbacks .................................................. 2
  2.3 Module declaration ............................................ 3
  2.4 Lua Attributes .................................................. 4
  2.5 Prefixed names for \texttt{luatex\textbackslash{}X} primitives ... 4

3 Implementation 4
  3.1 \texttt{luatexbase} interface .......................... 4
  3.2 Legacy \texttt{luatexbase} sub-packages .................. 12
  3.3 Legacy Lua code ................................................ 12

1 Overview

\LaTeXe\添加了一些引擎特定的函数到\LaTeXe\。支持这些功能的兼容层，现在可以在\LaTeXe\内核中提供，或者作为独立文件\texttt{ltluatex.tex}供非专业人士使用。这些功能是以前的\texttt{luatex}和\texttt{luatexbase}包由Heiko Oberdiek, Élie Roux, Manuel Pégourié-Gonnar和Philipp Gesang编写的。但是，这些接口并不完全相同。

这些被定义在本包中的接口是根据原来的\texttt{luatexbase}包进行的，提供了一个向后兼容层，以支持内核级的原语支持和使用\texttt{luatexbase}的现有代码。

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


2 The \texttt{luatexbase} package interface

2.1 Catcode tables

2.1.1 \TeX

\texttt{\CatcodeTableIniTeX} \TeX{} access to predefined catcode tables.

\texttt{\CatcodeTableString} The first four tables are aliases giving alternative names for some catcodetables that are defined in the \texttt{ltluatex} core.

\texttt{\CatcodeTableLaTeX} \texttt{\CatcodeTableOther} is like \texttt{\CatcodeTableString} except that the catcode of space is 12 (other).

\texttt{\CatcodeTableLaTeXAtLetter} \texttt{\CatcodeTableExpl} is similar to the environment set by the \texttt{\ExplSyntaxOn} note that this only affects catcode settings, not for example \texttt{\endlinechar}.

One difference between this implementation and the tables defined in the earlier \texttt{luatexbase} package is that these tables are defined to match the settings used by \texttt{\ETEX} over the full Unicode range (as set in the file \texttt{unicode-letters.def}).

\texttt{\SetCatcodeRange} An alias for \texttt{\SetRangeCatcode} which is defined in the \texttt{ctablestack} package imported into this version of \texttt{luatexbase}. (The order of arguments is the same despite the variation in the naming). This is useful for setting up a new catcode table and assigns a given catcode to a range of characters.

\texttt{\BeginCatcodeRegime} \texttt{\EndCatcodeRegime} A simple wrapper around \texttt{\@pushcatcodetable} providing a slightly different interface. The usage is:

\texttt{\BeginCatcodeRegime(catcode table) (code)}

\texttt{\EndCatcodeRegime}

\texttt{\PushCatcodeTableNumStack} \texttt{\PopCatcodeTableNumStack} These are defined to be aliases for \texttt{\@pushcatcodetable} and \texttt{\@popcatcodetable} although the actual implementation is quite different to the older packages, the use of the commands should match.

\texttt{\newluatexcatcodetable \setluatexcatcodetable} Aliases for the \texttt{\ltluatex} functions dropping \texttt{\luatex} from the name to match the convention of not using \texttt{\luatex}-prefixed names for the \texttt{\luatex}-\texttt{\TeX} primitives.

2.1.2 Lua

The standard way to access catcode table numbers from Lua in \texttt{ltluatex} is the \texttt{\registernumber} function. This package provides a \texttt{\catcodetables} table with a metatable that accesses this function and is extended with aliases for the predefined tables so you can use \texttt{\catcodetables.expl} as an alternative to \texttt{\catcodetables.CatcodeTableExpl}, both being equivalent to \texttt{\registernumber('CatcodeTableExpl')}.

2.2 Lua Callbacks

The \texttt{luatexbase} table is extended with some additional Lua functions to provide the interfaces provided by the previous implementation.

\footnote{This interface was previously defined in the \texttt{\luatexbase-cctbl} sub-package.}

\footnote{This interface was previously defined in the \texttt{\luatexbase-mcb} sub-package.}
priority_in_callback \langle name \rangle \langle description \rangle
This function was previously defined in the development sources of the previous implementation. Here it is defined as an alias for the function \texttt{in_callback} provided by \texttt{ltlatex}. Given a callback and a description string, it returns true if a callback function with that description is currently registered.

is_active_callback \langle name \rangle \langle description \rangle
This boolean function was defined in the development sources of the previous implementation. Here it is defined as an alias for the function \texttt{in_callback} provided by \texttt{ltlatex}. Given a callback and a description string, it returns true if a callback function with that description is currently registered.

reset_callback \langle name \rangle \langle make\_false \rangle
This function unregisters all functions registered for the callback \langle name \rangle. If \langle make\_false \rangle is true, the callback is then set to false (rather than nil). Unlike the earlier implementation this version does call \texttt{remove\_from\_callback} on each function in the callback list for \langle name \rangle, and each removal will be recorded in the log.

remove_from_callback \langle name \rangle \langle description \rangle
This function is unchanged from the kernel-level implementation. It is backward compatible with the previous \texttt{luatexbase} package but enhanced as it returns the removed callback and its description. Together with the \texttt{callback\_descriptions} function this allows much finer control over the order of functions in a callback list as the functions can be removed then re-added to the list in any desired order.

add_to_callback \langle name \rangle \langle function \rangle \langle description \rangle \langle priority \rangle
This function is defined as a wrapper around the kernel-level implementation, which does not have the fourth \langle priority \rangle argument.

If multiple callbacks are registered to a callback of type exclusive then \texttt{ltlatex} raises an error, but here it is allowed if priority is 1, in which case the \texttt{reset\_callback} is first called to remove the existing callback.

In general the priority argument is implemented by temporarily removing some callbacks from the list and replacing them after having added the new callback.

create_callback \langle name \rangle \langle type \rangle \langle default \rangle
This function is unchanged from kernel-level implementation, the only change is a change of terminology for the types of callback, the type \texttt{first} is now classified as exclusive and the kernel code raises an error if multiple callback functions are registered. The previous \texttt{luatexbase} implementation allowed multiple functions to be registered, but only activated the first in the list.

2.3 Module declaration

2.3.1 \TeX

\texttt{\RequireLuaModule \langle file \rangle [\langle info \rangle]}
This command is provided as a wrapper around \directlua{require(file)}, and executes the Lua code in the specified file. The optional argument is accepted but ignored.

Current versions of Lua\TeX{} all use the kpse \TeX{} path searching library with the require function, so the more complicated definition used in earlier implementations is no longer needed.

2.3.2 Lua

\texttt{provides\_module}\langle info\rangle

The \texttt{luatexbase} version of \texttt{provides\_module} returns a list of log and error functions so that it is usually called as:

\begin{verbatim}
local err, warning, info, log = luatexbase.provides_module({name=..}
\end{verbatim}

The returned functions are all instances of the functions provided by the kernel: \texttt{module\_error}, \texttt{module\_warning} and \texttt{module\_info}. They all use their first argument as a format string for any later arguments.

\texttt{errwarinf}\langle name\rangle

Returns four error and warning functions associated with \langle name\rangle mostly a helper function for \texttt{provides\_module}, but can be called separately.

2.4 Lua Attributes and Whatsits

2.4.1 \TeX{}

\begin{verbatim}
\newluatexattribute \setluatexattribute \unsetluatexattribute
\end{verbatim}

As for catcode tables, aliases for the attribute allocation functions are provided with \texttt{luatex} in the names.

2.4.2 Lua

The lua code in this section is concerned with an experimental whatsit handling suite of functions in the original package. This is not fully documented here and is guarded by the docstrip guard \texttt{whatst} so it may optionally be included or excluded from the sources when the package is built.

2.5 Prefixed names for lua\TeX{} primitives

\begin{verbatim}
\luatexattributedef \luatexcatcodetable
\luatexluascapestring \luatexlualatex
\luatexoutputbox \luatexscantextokens
\end{verbatim}

Aliases for commonly used lua\TeX{} primitives that existing packages using \texttt{luatexbase} use with prefixed names.

If additional primitives are required it is recommended that the code is updated to use unprefixed names. To ensure that the code works with the original \texttt{luatexbase} package on older formats you may use the lua function \texttt{tex.enableprimitives} to enable some or all primitives to be available with unprefixed names.

\footnote{This interface was previously defined in the \texttt{luatexbase-attr} sub-package.}

4
3 Implementation

3.1 luatexbase interface

\begin{verbatim}
(+emu)
\edef\emuatcatcode{\the\catcode`\@}
\catcode`\@=11

Load ctablestack.
\ifx@setrangecatcode@undefined
\ifx\RequirePackage@undefined
\input{ctablestack.sty}%
\else
\RequirePackage{ctablestack}
\fi
\else
\fi

Simple require wrapper as we now assume require implicitly uses the kpathsea search library.
\def\RequireLuaModule#1{\directlua{require("#1")}\@gobbleoptarg}

In \LaTeX{} (or plain macro package that has defined \texttt{@ifnextchar}) use \texttt{@ifnextchar} otherwise use a simple alternative, in practice this will never be followed by a brace group, so full version of \texttt{@ifnextchar} not needed.

\ifdefined\@ifnextchar
\def\@gobbleoptarg\[#1\]{%}
\else
\long\def\@gobbleoptarg#1{\ifx[#1\expandafter\@gobbleoptarg\fi#1}%
\fi

Extended catcode table support. Use the names from the previous luatexbase and luatex packages.
\let\CatcodeTableIniTeX\catcodetable@initex
\let\CatcodeTableString\catcodetable@string
\let\CatcodeTableLaTeX\catcodetable@latex
\let\CatcodeTableLaTeXAtLetter\catcodetable@atletter

Additional tables declared in the previous interface.
\newcatcodetable\CatcodeTableOther
\setcatcodetable\CatcodeTableOther{\%}
\catcodetable\CatcodeTableString
\catcode32 12 }
\newcatcodetable\CatcodeTableExpl
\setcatcodetable\CatcodeTableExpl{\%}
\catcodetable\CatcodeTableLaTeX
\catcode126 10 \% tilde is a space char
\catcode32 9 \% space is ignored
\catcode9 9 \% tab also ignored
\catcode95 11 \% underscore letter
\catcode58 11 \% colon letter
\end{verbatim}
Top level access to catcodetable stack.
\def\BeginCatcodeRegime#1{% 
\@pushcatcodetable 
\catcodetable#1\relax}
\def\EndCatcodeRegime{% 
\@popcatcodetable}

The implementation of the stack is completely different, but usage should match.
\let\PushCatcodeTableNumStack\@pushcatcodetable 
\let\PopCatcodeTableNumStack\@popcatcodetable

A simple copy.
\let\SetCatcodeRange\@setrangecatcode 
Another copy.
\let\setcatcodetable\@setcatcodetable

3.1.1 Additional lua code
\directlua{
Remove all registered callbacks, then disable. Set to false if optional second argument is true.
function luatexbase.reset_callback(name,make_false)
  for _,v in pairs(luatexbase.callback_descriptions(name))
do
  luatexbase.remove_from_callback(name,v)
end
  if make_false == true then
    luatexbase.disable_callback(name)
  end

  Allow exclusive callbacks to be over-written if priority argument is 1 to match the “first” semantics of the original package.
  First save the kernel function.
  luatexbase.base_add_to_callback=luatexbase.add_to_callback
  Implement the priority argument by taking off existing callbacks that have higher priority than the new one, adding the new one, Then putting the saved callbacks back.
function luatexbase.add_to_callback(name,fun,description,priority)
  local priority= priority
  if priority==nil then
    priority=\string#luatexbase.callback_descriptions(name)+1
  end
  if(\string#luatexbase.callbacktypes[name] == 3 and
    priority == 1 and
    \string#luatexbase.callback_descriptions(name)==1) then
    luatexbase.module_warning("luatexbase",
      "resetting exclusive callback: " .. name)
  luatexbase.reset_callback(name)
local saved_callback={},ff,dd
for k,v in pairs(luatexbase.callback_descriptions(name)) do
  if k >= priority then
    ff,dd= luatexbase.remove_from_callback(name, v)
    saved_callback[k]={ff,dd}
  end
end
luatexbase.base_add_to_callback(name,fun,description)
for k,v in pairs(saved_callback) do
  luatexbase.base_add_to_callback(name,v[1],v[2])
end
return
end

Emulate the catcodetables table. Explicitly fill the table rather than rely on
the metatable call to registernumber as that is unreliable on old LuaT\TeX.
luatexbase.catcodetables=setmetatable(
  {{['latex-package'] = \number\CatcodeTableLaTeXAtLetter,
  ini = \number\CatcodeTableIniTeX,
  string = \number\CatcodeTableString,
  other = \number\CatcodeTableOther,
  latex = \number\CatcodeTableLaTeX,
  expl = \number\CatcodeTableExpl,
  expl3 = \number\CatcodeTableExpl},
  { __index = function(t,key)
    return luatexbase.registernumber(key) or nil
  end})
)

On old LuaT\TeX workaroudn hashtable issues. Allocate in \TeX, and also di-
rectly add to luatexbase.catcodetables.
\directlua{luatexversion<80 %
  \def\newcatcodetable#1{%
    \e@alloc\catcodetable\chardef
    \e@alloc@ccodetable@count\m@ne\string8000\string#1%
    \initcatcodetable\allocationnumber
    \escapename\string8000\string#1%
    \directlua{luatexbase.catcodetables[\string#1]='%
    \the\allocationnumber}}%
  \fi\}

priority_in_callback returns position in the callback list. Not provided
by default by the kernel as usually it is just used as a boolean test, for which
in_callback is provided.

function luatexbase.priority_in_callback (name,description)
  for i,v in ipairs(luatexbase.callback_descriptions(name))
do
if v == description then
    return i
end

return false
end

The (unreleased) version 0.7 of luatexbase provided this boolean test under a
different name, so we provide an alias here.

luatexbase.is_active_callback = luatexbase.in_callback

ltlatex implementation of provides_module does not return print functions
so define modified version here.

luatexbase.base_provides_module=luatexbase.provides_module
function luatexbase.errwarinf(name)
    return
function(s,...) return luatexbase.module_error(name, s:format(...)) end,
function(s,...) return luatexbase.module_warning(name, s:format(...)) end,
function(s,...) return luatexbase.module_info(name, s:format(...)) end,
function(s,...) return luatexbase.module_info(name, s:format(...)) end
end

function luatexbase.provides_module(info)
    luatexbase.base_provides_module(info)
    return luatexbase.errwarinf(info.name)
end

Same for attribute table as catcode tables. In old LuaTeX, add to the
luatexbase.attributes table directly.

\ifnum\luatexversion<80 %
\edef\newattribute#1{\%\edef\attribute\attributedef\edef\attribute@count\m@ne\edef\top\string#1\%\edef\escapechar\m@ne\directlua{luatexbase.attributes[\"\string#1"]=%\the\allocationnumber}}%\fi

\edef\percentchar\@undefined
{\catcode\%=12 \gdef\percentchar{%}
\fi
\langle\whatsit\rangle
\directlua{
local copynode = node.copy
local newnode = node.new
local nodesubtype = node.subtype
local nodetype = node.id
local stringformat = string.format
local tableunpack = unpack or table.unpack
8
local texiowrite_nl = texio.write_nl
local texiowrite = texio.write
local whatsit_t = nodetype"whatsit"
local user_defined_t = nodetype"user_defined"
local unassociated = "__unassociated"
local user_whatsits = { __unassociated = { } }
local whatsit_ids = { }
local anonymous_whatsits = 0
local anonymous_prefix = "anon"

User whatsit allocation is split into two functions: new_user_whatsit_id registers a new id (an integer) and returns it. This is a wrapper around new_whatsit but with the extra package argument, and recording the mapping in lua tables

If no name given, generate a name from a counter.

local new_user_whatsit_id = function (name, package)
  if name then
    if not package then
      package = unassociated
    end
  else % anonymous
    anonymous_whatsits = anonymous_whatsits + 1
    package = unassociated
    name = anonymous_prefix .. tostring(anonymous_whatsits)
  end

  local whatsitdata = user_whatsits[package]
  if not whatsitdata then
    whatsitdata = { }
    user_whatsits[package] = whatsitdata
  end

  local id = whatsitdata[name]
  if id then %- warning
    warning("replacing whatsit \%s: \%s \%s", package, name, id)
  else %- new id
    id=luatexbase.new_whatsit(name)
    whatsitdata[name] = id
    whatsit_ids[id] = { name, package }
  end
  return id
end

luatexbase.new_user_whatsit_id = new_user_whatsit_id

new_user_whatsit first registers a new id and then also creates the corresponding whatsit node of subtype user-defined. Return a nullary function that delivers copies of the whatsit.

Alternatively, the first argument can be a whatsit node that will then be used
as prototype.

```lua
local new_user_whatsit = function (req, package)
    local id, whatsit
    if type(req) == "string" then
        id = new_user_whatsit_id(req, package)
        whatsit = newnode(whatsit_t, user_defined_t)
        whatsit.user_id = id
    elseif req.id == whatsit_t and req.subtype == user_defined_t then
        id = req.user_id
        whatsit = copynode(req)
        if not whatsit_ids[id] then
            warning("whatsit id \%d unregistered; " .. "inconsistencies may arise", id)
        end
    end
    return function () return copynode(whatsit) end, id
end
luatexbase.new_user_whatsit = new_user_whatsit
```

If one knows the name of a user whatsit, its corresponding id can be retrieved by means of `get_user_whatsit_id`.

```lua
local get_user_whatsit_id = function (name, package)
    if not package then
        package = unassociated
    end
    return user_whatsits[package][name]
end
luatexbase.get_user_whatsit_id = get_user_whatsit_id
```

The inverse lookup is also possible via `get_user_whatsit_name`.

```lua
local get_user_whatsit_name = function (asked)
    local id
    if type(asked) == "number" then
        id = asked
    elseif type(asked) == "function" then
        % node generator
        local n = asked()
        id = n.user_id
    else % node
        id = asked.user_id
    end
    local metadata = whatsit_ids[id]
    if not metadata then % unknown
        warning("whatsit id \%d unregistered; " .. "inconsistencies may arise", id)
        return "", ""
    end
    return tableunpack(metadata)
end
luatexbase.get_user_whatsit_name = get_user_whatsit_name
```

A function that outputs the current allocation status to the terminal.
local dump_registered_whatsits = function (asked_package)
  local whatsit_list = { }
  if asked_package then
    local whatsitdata = user_whatsits[asked_package]
    if not whatsitdata then
      error("(no user whatsits registered for package %s", asked_package)
      return
    end
    texiowrite_nl("(user whatsit allocation stats for " ..
      asked_package)
    for name, id in next, whatsitdata do
      whatsit_list[#whatsit_list+1] = stringformat("(%s:%s @\percentchar d)",
        asked_package, name, id)
    end
  else
    texiowrite_nl("(user whatsit allocation stats")
    texiowrite_nl(stringformat(" ((total @\percentchar d)\string\n      (anonymous @\percentchar d))", current_whatsit, anonymous_whatsits))
    for package, whatsitdata in next, user_whatsits do
      for name, id in next, whatsitdata do
        whatsit_list[#whatsit_list+1] = stringformat("(%s:%s @\percentchar d)",
          package, name, id)
      end
    end
  end
  texiowrite_nl" ("
  local first = true
  for i=1, #whatsit_list do
    if first then
      first = false
      else \% indent
        texiowrite_nl" "
    end
    texiowrite(whatsit_list[i])
  end
  texiowrite\string\n"
end
luatexbase.dump_registered_whatsits = dump_registered_whatsits

Lastly, we define a couple synonyms for convenience.
luatexbase.newattribute = new_attribute
luatexbase.newuserwhatsit = new_user_whatsit
luatexbase.newuserwhatsitid = new_user_whatsit_id
luatexbase.getuserwhatsitid = get_user_whatsit_id
luatexbase.getuserwhatsitname = get_user_whatsit_name
luatexbase.dumpregisteredwhatsits = dump_registered_whatsits
Resolve name clashes and prefixed name issues.

Top level \texttt{luatexbase} macros
\begin{verbatim}
\let\newluatexattribute\newattribute
\let\setluatexattribute\setattrtribute
\let\unsetluatexattribute\unsetattribute
\let\newluatexcatcodetable\newcatcodetable
\let\setluatexcatcodetable\setcatcodetable
\end{verbatim}

Internal \texttt{luatexbase} macros
\begin{verbatim}
\let\luatexbase@directlua\directlua
\let\luatexbase@ensure@primitive\@gobble
\end{verbatim}

\texttt{Lua\TeX} primitives
\begin{verbatim}
\let\luatexattribute\attribute
\let\luatexattributedef\attributedef
\let\luatexcatcodetable\catcodetable
\let\luatexluaescapestring\luaescapestring
\let\luatexlatexlua\latexlua
\let\luatexoutputbox\outputbox
\let\luatexscantextokens\scantextokens
\end{verbatim}

Reset catcode of \texttt{@}.
\begin{verbatim}
\catcode'@=\emuatcatcode\relax
\end{verbatim}

3.2 Legacy \texttt{luatexbase} sub-packages

The original \texttt{luatexbase} was comprised of seven sub packages that could in principle be loaded separately. Here we define them all with the same code that just loads the main package, they are distinguished just by the \texttt{\ProvidesPackage} specified above at the start of the file.
\begin{verbatim}
\ifx\RequirePackage\undefined
\input{luatexbase.sty}\
\else
\RequirePackage{luatexbase}
\fi
\end{verbatim}

3.3 Legacy Lua code

The original \texttt{luatexbase} included a file \texttt{luatexbase.loader.lua} that could be loaded independently of the rest of the package. This really doesn’t need to do anything!
\begin{verbatim}
\ifx\ lua-lua
\.luatexbase = \luatexbase or { }
\fi
\end{verbatim}