luatexbase.dtx
(LuaTEX-specific support, luatexbase interface)

David Carlisle and Joseph Wright*

2015/10/04

Contents

1 Overview

LuaTEX adds a number of engine-specific functions to TEX. Support for those is now available for this area in the LATEX kernel and as an equivalent standalone file `ltluatex.tex` for plain users. The functionality there is derived from the earlier luatex and luatexbase packages by Heiko Oberdiek, Élie Roux, Manuel Pégourié-Gonnar and Philipp Gesang. However, the interfaces are not all identical.

The interfaces defined in this package are closely modelled on the original luatexbase package, and provide a compatibility layer between the new kernel-level support and existing code using luatexbase.

---

*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.

2 The **luatexbase** package interface

2.1 Catcode tables[1]

2.1.1 TEX

`\CatcodeTableIniTeX`
`\CatcodeTableString`
`\CatcodeTableLaTeX`
`\CatcodeTableLaTeXAtLetter`
`\CatcodeTableOther`
`\CatcodeTableExpl`

TEX access to predefined catcode tables.

The first four tables are aliases giving alternative names for some catcodetables that are defined in the ltluatex core.

`\CatcodeTableOther` is like `\CatcodeTableString` except that the catcode of space is 12 (other).

`\CatcodeTableExpl` is similar to the environment set by the expl3 command `\ExplSyntaxOn` note that this only affects catcode settings, not for example `\endlinechar`.

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

`\SetCatcodeRange`

An alias for `\@setrangecatcode` which is defined in the ctablestack package imported into this version of 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.

`\BeginCatcodeRegime`
`\EndCatcodeRegime`

A simple wrapper around `\@pushcatcodetable` providing a slightly different interface. The usage is:

`\BeginCatcodeRegime`⟨*catcode table*⟩
    ⟨*code*⟩
`\EndCatcodeRegime`

`\PushCatcodeTableNumStack`
`\PopCatcodeTableNumStack`

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

`\newluatexcatcodetable`
`\setluatexcatcodetable`

Aliases for the ltluatex functions dropping `luatex` from the name to match the convention of not using `luatex`-prefixed names for the LuaTEX primitives.

2.1.2 Lua

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

2.2 Lua Callbacks[2]

The `luatexbase` table is extended with some additional Lua functions to provide the interfaces provided by the previous implementation.

---

[1] This interface was previously defined in the luatexbase-cctbl sub-package.

[2] This interface was previously defined in the luatexbase-mcb sub-package.

`priority_in_callback` ⟨*name*⟩⟨*description*⟩
As in the earlier interfaces the function is provided to return a number indicating the position of a specified function in a callback list. However it is usually used just as a boolean test that the function is registered with the callback. Kernel-level support does not directly expose the priority numbers, however the function here is defined to return the number of the specified function in the list returned by `luatexbase.callback_descriptions`.

`is_active_callback` ⟨*name*⟩⟨*description*⟩
This boolean function was defined in the development sources of the previous implementation. Here it is defined as an alias for the function `in_callback` provided by ltluatex. Given a callback and a description string, it returns true if a callback function with that description is currently registered.

`reset_callback` ⟨*name*⟩⟨*make_false*⟩
This function unregisters all functions registered for the callback ⟨*name*⟩. If ⟨*make_false*⟩ is true, the callback is then set to false (rather than nil). Unlike the earlier implementation This version does call `remove_from_callback` on each function in the callback list for ⟨*name*⟩, and each removal will be recorded in the log.

`remove_from_callback` ⟨*name*⟩⟨*description*⟩
This function is unchanged from the kernel-level implementation. It is backward compatible with the previous luatexbase package but enhanced as it returns the removed callback and its description. Together with the `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` ⟨*name*⟩⟨*function*⟩⟨*description*⟩⟨*priority*⟩
This function is defined as a wrapper around the kernel-level implementation, which does not have the fourth ⟨*priority*⟩ argument.

If multiple callbacks are registered to a callback of type exclusive then ltluatex raises an error, but here it is allowed if `priority` is 1, in which case the `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` ⟨*name*⟩⟨*type*⟩⟨*default*⟩
This function is unchanged from kernel-level implementation, the only change is a change of terminology for the types of callback, the type `first` is now classified as `exclusive` and the kernel code raises an error if multiple callback functions are registered. The previous luatexbase implementation allowed multiple functions to be registered, but only activated the first in the list.

2.3 Module declaration[3]

2.3.1 TeX

`\RequireLuaModule` ⟨*file*⟩ `[`⟨*info*⟩`]`

---

[3] This interface was previously defined in the luatexbase-modutils sub-package.

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 LuaTEX 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

`provides_module` ⟨*info*⟩

The luatexbase version of `provides_module` returns a list of log and error functions so that it is usually called as:

`local err, warning, info, log = luatexbase.provides_module({name=..`

The returned functions are all instances of the functions provided by the kernel: `module_error`, `module_warning` and `module_info`, They all use their first argument as a format string fo rany later arguments.

`errwarinf` ⟨*name*⟩

Returns four error and warning functions associated with ⟨*name*⟩ mostly a helper function for `provides\_module`, but can be called separately.

2.4 Lua Attributes and Whatsits[4]

2.4.1 TEX

`\newluatexattribute`
`\setluatexattribute`
`\unsetluatexattribute`

As for catcode tables, aliases for the attribute allocation functions are provided with `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 guraded by the docstrip guard `whatsit` so it may optionally be included or excluded from the sources when the package is built.

2.5 Prefixed names for luaTEX primitives

`\luatexattributedef`
`\luatexcatcodetable`
`\luatexluaescapestring`
`\luatexlatelua`
`\luatexoutputbox`
`\luatexscantextokens`

Aliases for commonly ued luaTEX primitives that existing packages using luatexbase use with prefixed names.

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

---

[4] This interface was previously defined in the luatexbase-attr sub-package.

3 Implementation

3.1 luatexbase interface

```
⟨*emu⟩
\edef\emuatcatcode{\the\catcode`\@}
\catcode`\@=11
```

Load `ctablestack`.

```
\ifx\@setrangecatcode\@undefined
  \ifx\RequirePackage\@undefined
    \input{ctablestack.sty}%
  \else
    \RequirePackage{ctablestack}
  \fi
\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 `\@ifnextchar`) use `\@ifnextchar` otherwise use a simple alternative, in practice this will never be followed by a brace group, so full version of `\@ifnextchar` not needed.

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

\def\@gobble@optarg[#1]{}
```

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
}
```

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
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(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)
  end
  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 LuaTeX.

```
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 LuaTeX workaround hashtable issues. Allocate in TeX, and also directly add to `luatexbase.catcodetables`.

```
\ifnum\luatexversion<80 %
\def\newcatcodetable#1{%
  \e@alloc\catcodetable\chardef
    \e@alloc@ccodetable@count\m@ne{"8000}#1%
  \initcatcodetable\allocationnumber
  {\escapechar=\m@ne
  \directlua{luatexbase.catcodetables['\string#1']=%
    \the\allocationnumber}}%
}
\fi
\directlua{
```

`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
  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
```

ltluatex 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 %
\def\newattribute#1{%
  \e@alloc\attribute\attributedef
    \e@alloc@attribute@count\m@ne\e@alloc@top#1%
  {\escapechar=\m@ne
  \directlua{luatexbase.attributes['\string#1']=%
    \the\allocationnumber}}%
}
\fi
```

Define a safe percent command for plain TeX.

```
\ifx\@percentchar\@undefined
  {\catcode`\%=12 \gdef\@percentchar{%}}
\fi
```

```
⟨*whatsit⟩
\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
local texiowrite_nl     = texio.write_nl
local texiowrite        = texio.write
local whatsit_t         = nodetype"whatsit"
local user_defined_t    = nodesubtype"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
        warning("defining anonymous user whatsit no. \@percentchar
                  d", anonymous_whatsits)
        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 \@percentchar s:\@percentchar
                  s (\@percentchar d)", 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.

```
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 \@percentchar 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`.

```
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`.

```
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 \@percentchar 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
                     \@percentchar s)", asked_package)
            return
        end
        texiowrite_nl("(user whatsit allocation stats for " ..
                          asked_package)
        for name, id in next, whatsitdata do
            whatsit_list[\string#whatsit_list+1] =
                stringformat("(\@percentchar s:\@percentchar
                     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[\string#whatsit_list+1] =
                    stringformat("(\@percentchar s:\@percentchar
                        s \@percentchar d)", package, name, id)
            end
        end
    end
    texiowrite_nl" ("
    local first = true
    for i=1, \string#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
```

```
}
⟨/whatsit⟩
```

Resolve name clashes and prefixed name issues.

Top level luatexbase macros

```
\let\newluatexattribute\newattribute
\let\setluatexattribute\setattribute
\let\unsetluatexattribute\unsetattribute
\let\newluatexcatcodetable\newcatcodetable
\let\setluatexcatcodetable\setcatcodetable
```

Internal luatexbase macros

```
\let\luatexbase@directlua\directlua
\let\luatexbase@ensure@primitive\@gobble
```

LuaTEX primitives

```
\let\luatexattribute\attribute
\let\luatexattributedef\attributedef
\let\luatexcatcodetable\catcodetable
\let\luatexluaescapestring\luaescapestring
\let\luatexlatelua\latelua
\let\luatexoutputbox\outputbox
\let\luatexscantextokens\scantextokens
```

Reset catcode of `@`.

```
\catcode`\@=\emuatcatcode\relax
⟨/emu⟩
```

3.2 Legacy luatexbase sub-packages

The original 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 `\ProvidesPackage` specified above at the start of the file.

```
⟨*emu-cmp, emu-mod, emu-loa, emu-reg, emu-att, emu-cct, emu-mcb⟩
\ifx\RequirePackage\undefined
  \input{luatexbase.sty}%
\else
  \RequirePackage{luatexbase}
\fi
⟨/emu-cmp, emu-mod, emu-loa, emu-reg, emu-att, emu-cct, emu-mcb⟩
```

3.3 Legacy Lua code

The original luatexbase included a file `luatexbase.loader.lua` that could be loaded independently of the rest of the package. This really doesn't need to do anything!

```
⟨*emu-lua⟩
luatexbase = luatexbase or { }
⟨/emu-lua⟩
```