

The luatexbase-mcb package

Manuel Pégourié-Gonnard
mpg@elzevir.fr

Élie Roux
elie.roux@telecom-bretagne.eu

2010/05/27 v0.2a

Abstract

This package manages the callback adding and removing, by adding `callback.add` and `callback.remove`, and overwriting `callback.register`. It also allows to create and call new callbacks. For an introduction on this package (among others), please refer to the document `luatextra-reference.pdf`.

Warning. Currently assumes that `luatexbase-modutils` has been previously loaded. (This is a temporary limitation.)

Contents

1	Documentation	1
2	Implementation	2
2.1	TeX package	2
2.1.1	Preliminaries	2
2.1.2	Load supporting Lua module	4
2.2	Lua module	4
2.2.1	Module identification	4
2.2.2	Initialisations	4
2.2.3	Unsorted stuff	6
2.2.4	Public functions	8
3	Test files	12

1 Documentation

LuaTeX provides an extremely interesting feature, named callbacks. It allows to call some lua functions at some points of the TeX algorithm (a *callback*), like when TeX breaks lines, puts vertical spaces, etc. The LuaTeX core offers a function called `callback.register` that enables to register a function in a callback.

The problem with `callback.register` is that it registers only one function in a callback. For a lot of callbacks it can be common to have several packages registering their function in a callback, and thus it is impossible with them to be compatible with each other.

This package solves this problem by adding mainly one new function `callback`. `add` that adds a function in a callback. With this function it is possible for packages to register their function in a callback without overwriting the functions of the other packages.

The functions are called in a certain order, and when a package registers a callback it can assign a priority to its function. Conflicts can still remain even with the priority mechanism, for example in the case where two packages want to have the highest priority. In these cases the packages have to solve the conflicts themselves.

This package also provides a way to create and call new callbacks, in addition to the default LuaTeX callbacks.

Limitations

This package only works for callbacks where it's safe to add multiple functions without changing the functions' signatures. There are callbacks, though, where registering several functions is not possible without changing the function's signatures, like for example the readers callbacks. These callbacks take a filename and give the datas in it. One solution would be to change the functions' signature to open it when the function is the first, and to take the datas and modify them eventually if they are called after the first. But it seems rather fragile and useless, so it's not implemented. With these callbacks, in this package we simply execute the first function in the list.

Other callbacks in this case are `define_font` and `open_read_file`. There is though a solution for several packages to use these callbacks, see the implementation of `luatextra`.

2 Implementation

2.1 T_EX package

```
1 < *texpackage >
```

2.1.1 Preliminaries

Reload protection, especially for Plain T_EX.

```
2 \csname lltxb@mcb@loaded\endcsname
3 \expandafter\let\csname lltxb@mcb@loaded\endcsname\endinput
```

Catcode defenses.

```
4 \begingroup
5 \catcode123 1 % {
6 \catcode125 2 % }
7 \catcode 35 6 % #
8 \toks0{}%
9 \def\x{}%
10 \def\y#1 #2 {%
11 \toks0\expandafter{\the\toks0 \catcode#1 \the\catcode#1}%
12 \edef\x{\x \catcode#1 #2}}%
13 \y 123 1 % {
14 \y 125 2 % }
15 \y 35 6 % #
16 \y 10 12 % ^^J
```

```

17 \y 34 12 % "
18 \y 36 3 % $ $
19 \y 39 12 % '
20 \y 40 12 % (
21 \y 41 12 % )
22 \y 42 12 % *
23 \y 43 12 % +
24 \y 44 12 % ,
25 \y 45 12 % -
26 \y 46 12 % .
27 \y 47 12 % /
28 \y 60 12 % <
29 \y 61 12 % =
30 \y 64 11 % @ (letter)
31 \y 62 12 % >
32 \y 95 12 % _ (other)
33 \y 96 12 % '
34 \edef\y#1{\endgroup\edef#1{\the\toks0\relax}\x}%
35 \expandafter\y\csname lltxb@mcb@AtEnd\endcsname

```

Package declaration.

```

36 \begingroup
37 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
38   \def\x#1[#2]{\immediate\write16{Package: #1 #2}}
39 \else
40   \let\x\ProvidesPackage
41 \fi
42 \expandafter\endgroup
43 \x{luatexbase-mcb}[2010/05/27 v0.2a Callback management for LuaTeX]

```

Make sure LuaTeX is used.

```

44 \begingroup\expandafter\expandafter\expandafter\endgroup
45 \expandafter\ifx\csname RequirePackage\endcsname\relax
46   \input ifluatex.sty
47 \else
48   \RequirePackage{ifluatex}
49 \fi
50 \ifluatex\else
51   \begingroup
52     \expandafter\ifx\csname PackageWarningNoLine\endcsname\relax
53       \def\x#1#2{\begingroup\newlinechar10
54         \immediate\write16{Package #1 warning: #2}\endgroup}
55     \else
56       \let\x\PackageWarningNoLine
57     \fi
58   \expandafter\endgroup
59   \x{luatexbase-mcb}{LuaTeX is required for this package. Aborting.}
60   \lltxb@mcb@AtEnd
61   \expandafter\endinput
62 \fi

```

2.1.2 Load supporting Lua module

First load luatexbase-loader (hence luatexbase-compat), then the supporting Lua module.

```
63 \begingroup\expandafter\expandafter\expandafter\endgroup
64 \expandafter\ifx\csname RequirePackage\endcsname\relax
65   \input luatexbase-modutils.sty
66 \else
67   \RequirePackage{luatexbase-modutils}
68 \fi
69 \luatexbase@directlua{require('luatexbase.mcb')}

    That's all folks!

70 \lltxb@mcb@AtEnd
71 \endpackage
```

2.2 Lua module

```
72  $\langle$ *lua $\rangle$ 
```

2.2.1 Module identification

```
73 module('luatexbase', package.seeall)
74 luatexbase.provides_module({
75   name       = "luamcallbacks",
76   version    = 0.2,
77   date       = "2010/05/12",
78   description = "register several functions in a callback",
79   author     = "Hans Hagen, Elie Roux and Manuel PÃlgourie-Gonnard",
80   copyright  = "Hans Hagen, Elie Roux and Manuel PÃlgourie-Gonnard",
81   license    = "CC0",
82 })
```

Shortcuts for error functions.

```
83 local log = log or function(...)
84   luatexbase.module_log('luamcallbacks', string.format(...))
85 end
86 local info = info or function(...)
87   luatexbase.module_info('luamcallbacks', string.format(...))
88 end
89 local warning = warning or function(...)
90   luatexbase.module_warning('luamcallbacks', string.format(...))
91 end
92 local err = err or function(...)
93   luatexbase.module_error('luamcallbacks', string.format(...))
94 end
```

2.2.2 Initialisations

callbacklist is the main list, that contains the callbacks as keys and a table of the registered functions a values.

```
95 local callbacklist = callbacklist or { }
```

A table with the default functions of the created callbacks. See `create` for further informations.

```
96 local lua_callbacks_defaults = { }
```

There are 4 types of callback:

- the ones taking a list of nodes and returning a boolean and eventually a new head (**list**)
- the ones taking datas and returning the modified ones (**data**)
- the ones that can't have multiple functions registered in them (**first**)
- the ones for functions that don't return anything (**simple**)

```
97 local list = 1
```

```
98 local data = 2
```

```
99 local first = 3
```

```
100 local simple = 4
```

`callbacktypes` is the list that contains the callbacks as keys and the type (list or data) as values.

```
101 local callbacktypes = callbacktypes or {
```

```
102   buildpage_filter = simple,
```

```
103   token_filter = first,
```

```
104   pre_output_filter = list,
```

```
105   hpack_filter = list,
```

```
106   process_input_buffer = data,
```

```
107   mlist_to_hlist = list,
```

```
108   vpack_filter = list,
```

```
109   define_font = first,
```

```
110   open_read_file = first,
```

```
111   linebreak_filter = list,
```

```
112   post_linebreak_filter = list,
```

```
113   pre_linebreak_filter = list,
```

```
114   start_page_number = simple,
```

```
115   stop_page_number = simple,
```

```
116   start_run = simple,
```

```
117   show_error_hook = simple,
```

```
118   stop_run = simple,
```

```
119   hyphenate = simple,
```

```
120   ligaturing = simple,
```

```
121   kerning = data,
```

```
122   find_write_file = first,
```

```
123   find_read_file = first,
```

```
124   find_vf_file = data,
```

```
125   find_map_file = data,
```

```
126   find_format_file = data,
```

```
127   find_opentype_file = data,
```

```
128   find_output_file = data,
```

```
129   find_truetype_file = data,
```

```
130   find_type1_file = data,
```

```
131   find_data_file = data,
```

```
132   find_pk_file = data,
```

```

133 find_font_file = data,
134 find_image_file = data,
135 find_ocr_file = data,
136 find_sfd_file = data,
137 find_enc_file = data,
138 read_sfd_file = first,
139 read_map_file = first,
140 read_pk_file = first,
141 read_enc_file = first,
142 read_vf_file = first,
143 read_ocr_file = first,
144 read_opentype_file = first,
145 read_truetype_file = first,
146 read_font_file = first,
147 read_type1_file = first,
148 read_data_file = first,
149 }

```

In LuaTeX version 0.43, a new callback called `process_output_buffer` appeared, so we enable it. Test the version using the `compat` package for, well, compatibility.

```

150 if luatexbase.luatexversion > 42 then
151     callbacktypes["process_output_buffer"] = data
152 end

```

As we overwrite `callback.register`, we save it as `internalregister`.

```

153 local internalregister = internalregister or callback.register

```

2.2.3 Unsorted stuff

A simple function we'll use later to understand the arguments of the `create` function. It takes a string and returns the type corresponding to the string or nil.

```

154 local function str_to_type(str)
155     if str == 'list' then
156         return list
157     elseif str == 'data' then
158         return data
159     elseif str == 'first' then
160         return first
161     elseif str == 'simple' then
162         return simple
163     else
164         return nil
165     end
166 end

```

This function and the following ones are only internal. This one is the handler for the first type of callbacks: the ones that take a list head and return true, false, or a new list head.

```

167 -- local
168 function listhandler (name)
169     return function(head,...)

```

```

170     local l = callbacklist[name]
171     if l then
172         local done = true
173         for _, f in ipairs(l) do
174             -- the returned value is either true or a new head plus true
175             rtv1, rtv2 = f.func(head,...)
176             if type(rtv1) == 'boolean' then
177                 done = rtv1
178             elseif type(rtv1) == 'userdata' then
179                 head = rtv1
180             end
181             if type(rtv2) == 'boolean' then
182                 done = rtv2
183             elseif type(rtv2) == 'userdata' then
184                 head = rtv2
185             end
186             if done == false then
187                 err("function \"%s\" returned false in callback '%s'",
188                     f.description, name)
189             end
190         end
191         return head, done
192     else
193         return head, false
194     end
195 end
196 end

```

The handler for callbacks taking datas and returning modified ones.

```

197 local function datahandler (name)
198     return function(data,...)
199         local l = callbacklist[name]
200         if l then
201             for _, f in ipairs(l) do
202                 data = f.func(data,...)
203             end
204         end
205         return data
206     end
207 end

```

This function is for the handlers that don't support more than one functions in them. In this case we only call the first function of the list.

```

208 local function firsthandler (name)
209     return function(...)
210         local l = callbacklist[name]
211         if l then
212             local f = l[1].func
213             return f(...)
214         else

```

```

215         return nil, false
216     end
217 end
218 end

```

Handler for simple functions that don't return anything.

```

219 local function simplehandler (name)
220     return function(...)
221         local l = callbacklist[name]
222         if l then
223             for _, f in ipairs(l) do
224                 f.func(...)
225             end
226         end
227     end
228 end

```

2.2.4 Public functions

The main function. The signature is `add (name, func, description, priority)` with `name` being the name of the callback in which the function is added; `func` is the added function; `description` is a small character string describing the function, and `priority` an optional argument describing the priority the function will have.

The functions for a callbacks are added in a list (in `callbacklist` `.callbackname`). If they have no priority or a high priority number, they will be added at the end of the list, and will be called after the others. If they have a low priority number, they will be added at the beginning of the list and will be called before the others.

Something that must be made clear, is that there is absolutely no solution for packages conflicts: if two packages want the top priority on a certain callback, they will have to decide the priority they will give to their function themselves. Most of the time, the priority is not needed.

```

229 function add_to_callback (name,func,description,priority)
230     if type(func) ~= "function" then
231         err("unable to add function, no proper function passed")
232         return
233     end
234     if not name or name == "" then
235         err("unable to add function, no proper callback name passed")
236         return
237     elseif not callbacktypes[name] then
238         err("unable to add function, '%s' is not a valid callback", name)
239         return
240     end
241     if not description or description == "" then
242         err("unable to add function to '%s', no proper description passed",
243             name)
244         return
245     end
246     if priority_in_callback(name, description) ~= 0 then
247         warning("function '%s' already registered in callback '%s'",

```



```

248         description, name)
249     end
250     local l = callbacklist[name]
251     if not l then
252         l = {}
253         callbacklist[name] = l
254         if not lua_callbacks_defaults[name] then
255             if callbacktypes[name] == list then
256                 internalregister(name, listhandler(name))
257             elseif callbacktypes[name] == data then
258                 internalregister(name, datahandler(name))
259             elseif callbacktypes[name] == simple then
260                 internalregister(name, simplehandler(name))
261             elseif callbacktypes[name] == first then
262                 internalregister(name, firsthandler(name))
263             else
264                 err("unknown callback type")
265             end
266         end
267     end
268     local f = {
269         func = func,
270         description = description,
271     }
272     priority = tonumber(priority)
273     if not priority or priority > #l then
274         priority = #l+1
275     elseif priority < 1 then
276         priority = 1
277     end
278     if callbacktypes[name] == first and (priority ~= 1 or #l ~= 0) then
279         warning("several callbacks registered in callback '%s', "
280             .."only the first function will be active.", name)
281     end
282     table.insert(l,priority,f)
283     log("inserting function '%s' at position %s in callback list for '%s'",
284         description, priority, name)
285 end

```

The function that removes a function from a callback. The signature is `remove (name, description)` with `name` being the name of callbacks, and `description` the description passed to `add`.

```

286 function remove_from_callback (name, description)
287     if not name or name == "" then
288         err("unable to remove function, no proper callback name passed")
289         return
290     elseif not callbacktypes[name] then
291         err("unable to remove function, '%s' is not a valid callback", name)
292         return
293     end

```

```

294     if not description or description == "" then
295         err(
296             "unable to remove function from '%s', no proper description passed",
297             name)
298         return
299     end
300     local l = callbacklist[name]
301     if not l then
302         err("no callback list for '%s'",name)
303         return
304     end
305     for k,v in ipairs(l) do
306         if v.description == description then
307             table.remove(l,k)
308             log("removing function '%s' from '%s'",description,name)
309             if not next(l) then
310                 callbacklist[name] = nil
311                 if not lua_callbacks_defaults[name] then
312                     internalregister(name, nil)
313                 end
314             end
315             return
316         end
317     end
318     warning("unable to remove function '%s' from '%s'",description,name)
319 end

```

This function removes all the functions registered in a callback.

```

320 function reset_callback (name)
321     if not name or name == "" then
322         err("unable to reset, no proper callback name passed")
323         return
324     elseif not callbacktypes[name] then
325         err("reset error, '%s' is not a valid callback", name)
326         return
327     end
328     if not lua_callbacks_defaults[name] then
329         internalregister(name, nil)
330     end
331     local l = callbacklist[name]
332     if l then
333         log("resetting callback list '%s'",name)
334         callbacklist[name] = nil
335     end
336 end

```

This first function creates a new callback. The signature is `create(name, ctype, default)` where **name** is the name of the new callback to create, **ctype** is the type of callback, and **default** is the default function to call if no function is registered in this callback.

The created callback will behave the same way LuaTeX callbacks do, you can add and remove functions in it. The difference is that the callback is not automatically called, the package developer

creating a new callback must also call it, see next function.

```
337 function create_callback(name, ctype, default)
338     if not name then
339         err("unable to call callback, no proper name passed", name)
340         return nil
341     end
342     if not ctype or not default then
343         err("unable to create callback '%s': "
344             .."callbacktype or default function not specified", name)
345         return nil
346     end
347     if callbacktypes[name] then
348         err("unable to create callback '%s', callback already exists", name)
349         return nil
350     end
351     local temp = str_to_type(ctype)
352     if not temp then
353         err("unable to create callback '%s', type '%s' undefined", name, ctype)
354         return nil
355     end
356     ctype = temp
357     lua_callbacks_defaults[name] = default
358     callbacktypes[name] = ctype
359 end
```

This function calls a callback. It can only call a callback created by the `create` function.

```
360 function call_callback(name, ...)
361     if not name then
362         err("unable to call callback, no proper name passed", name)
363         return nil
364     end
365     if not lua_callbacks_defaults[name] then
366         err("unable to call lua callback '%s', unknown callback", name)
367         return nil
368     end
369     local l = callbacklist[name]
370     local f
371     if not l then
372         f = lua_callbacks_defaults[name]
373     else
374         if callbacktypes[name] == list then
375             f = listhandler(name)
376         elseif callbacktypes[name] == data then
377             f = datahandler(name)
378         elseif callbacktypes[name] == simple then
379             f = simplehandler(name)
380         elseif callbacktypes[name] == first then
381             f = firsthandler(name)
382         else
383             err("unknown callback type")

```

```

384         end
385     end
386     return f(...)
387 end

```

This function tells if a function has already been registered in a callback, and gives its current priority. The arguments are the name of the callback and the description of the function. If it has already been registered, it gives its priority, and if not it returns false.

```

388 function priority_in_callback (name, description)
389     if not name or name == ""
390         or not callbacktypes[name]
391         or not description then
392         return 0
393     end
394     local l = callbacklist[name]
395     if not l then return 0 end
396     for p, f in pairs(l) do
397         if f.description == description then
398             return p
399         end
400     end
401     return 0
402 end

```

Finally we overwrite `callback.register` so that it outputs an error.

```

403 callback.register = function ()
404 err("function callback.register has been deleted by luamcallbacks, "
405 .."please use callback.add instead.")
406 end

```

That's all folks!

```

407 </lua>

```

3 Test files

A few basic tests for Plain and LaTeX.

```

408 <testplain>\input luatexbase-mcb.sty
409 <testlatex>\RequirePackage{luatexbase-mcb}
410 <*testplain,testlatex>
411 \catcode 64 11
412 \luatexbase@directlua{
413     local function one(head,...)
414         texio.write_nl("I'm number 1")
415         return head, true
416     end
417
418     local function two(head,...)
419         texio.write_nl("I'm number 2")
420         return head, true

```

```

421 end
422
423 local function three(head,...)
424     texio.write_nl("I'm number 3")
425     return head, true
426 end
427
428 luatexbase.add_to_callback("hpack_filter",one,"my sample function one",1)
429 luatexbase.add_to_callback("hpack_filter",two,"my sample function two",2)
430 luatexbase.add_to_callback("hpack_filter",three,"my sample function three",1)
431
432 luatexbase.remove_from_callback("hpack_filter","my sample function three")
433 }
434 </testplain,testlatex>
435 <testplain>\bye
436 <testlatex>\stop

```