LuaTEXtra references

Elie Roux
elie.roux@telecom-bretagne.eu

May 10, 2010

Preamble: What is this document?

This document describes, from the user or developer point of view, how to use the coherent set of packages luatextra, luainputenc, luaotfload, luamcallbacks and lualibs (previously luaextra.

If you are looking for documentation on the LuaTEX engine, please refer to the file `luatexref-t.pdf`, but note that using these packages will slightly modify some LuaTEX behaviours, especially the `callback` library.

If you are looking for technical details about one of the packages, you can refer to its documentation. Each of them have a complete documentation in the file `package.pdf` with package being the name of the package.

Contents

1 Introduction

Before going any further, we need to explain the difference between an engine, a format, and a program. If you know the difference between these, please go to section 1.3.

1.1 What is LuaTEX?

LuaTEX is a TEX engine. A TEX engine is a binary executable that provides some very low-level primitives, for example `\count` to set a counter to a certain value. Examples of engines are the old TEX 82, ε-TEX, pdfTEX, Omega, Aleph, LuaTEX and XETEX.

There are two main differences between LuaTEX and its predecessor, pdfTeX: LuaTEX now truely understands UTF-8 and enables the execution of some lua code at certain points of the TEX algorithm. These two features allow, among other things, to open modern fonts like OpenType or TrueType fonts.

1.2 What is the difference with LATEX?

The nature of LATEX is fundamentally different: LATEX is a TEX format. Examples of formats are Plain, LATEX and ConTEXt.

This distinction is very hard to make, first because you cannot find any documentation on it, and secondly because the only things everyone knows are programs. A program is a command you type in your terminal that calls an engine with a format. Here are the different things behind what you type:

- `tex` is the engine TEX 82 with the format Plain. Both were created by Donald Knuth a very long time ago.
- `pdftex` is the engine pdfTEX (in PDF mode) with the format Plain.
- `luatex` is the engine LuaTEX (in PDF mode) with the format Plain.
- `xetex` is the engine XETEX (in PDF mode) with the format Plain.
- `latex` is the engine pdfTEX (in DVI mode) with the format LATEX.
- `pdflatex` is the engine pdfTEX (in PDF mode) with the format LATEX.
- `lualatex` is the engine LuaTEX (in PDF mode) with the format LATEX.
- `xelatex` is the engine XETEX (in PDF mode) with the format LATEX.

pdfTEX and LuaTEX can produce both DVI and PDF output. To choose it, you can simply set `\pdfoutput` to 0 or 1.

As you can see, there is absolutely no corellation between LuaTEX and LATEX, even if the names are close. . .

1.3 What is LuaTEXtra?

LuaTEXtra is the adaptation of the engine LuaTEX to the formats Plain and LATEX. It enables these formats to benefit from the LuaTEX new features, like UTF-8 reading, attributes handling, extended registers, modern font opening, etc.

LuaTEX provides only very low-level macros for these features, and a lot of code has to be done by the format to really provide a good user experience.

With what I just said, a good question would be the following: why isn't luatextra integrated into LATEX and Plain?

The answer is simple: because Plain and LATEX are frozen. This means that they don't accept new code anymore, even if it's totally necessary to benefit from new engines. Thus, the only solution is to provide a package that every user will have to load.

1.4 Why is it necessary?

LuaTEXtra is necessary for users and developers because it provides a way for several packages to access ressources.

Let's take a very concrete example: LuaTEX has a totally new concept named attributes; basically they can be considered as special counts: you can name them with the primitive `\attribute`, like you can name an counter with `\count`, and you can give them a value with `\attributedef`, that works like `\countdef` (see the LuaTEX documentation for more details about attributes). Attributes, like counts, are described by numbers, for example you have `\attribute0`, `\attribute1`, etc.

Plain and LATEX provide a macro `\newcount` that takes the first free counter and attributes it a name, for example we can image that `\newcount\foo` will in fact do `\count25\foo`. This enables packages to automatically take a free counter. Without it, package developers would have to maintain a list saying "please don't take attribute number foo, it's for the package bar!".

Plain and LATEX do not provide `\newattribute` to allocate a free attribute; but we do it in luatextra. Without LuaTEXtra, people have to take arbitrary attributes and cross their fingers hoping that noone else will use the attributes they've chosen. With LuaTEXtra, you can safely take attributes, being sure that noone else will use them.

Another similar problem is callback access: by default LuaTEX does not provide a way to register several functions in a callback, so if two packages want to register a function in a callback, they won't be compatible, as only one will work. luamcallback provides a new mechanism for callbacks that enables this, allowing more package compatibility.

These two problems are very concrete, that's the main reason why LuaTEXtra has been created: without it a lot of package compatibility problems would have been raised.

1.5 The different packages

The LuaTEXtra coherent set of packages contains:

- luatextra: the basic general macros and lua functions

- luainputenc: adaptation of inputenc to LuaTeX
- luamcallbacks: pure lua package to be able to register several functions in a callback
- lualibs: additional lua functions (previously luaextra).

1.6 Overview

The main idea, as explained above, is to provide a safe way to use different packages that are not coherent, without loading several times the same code, or taking ressources (like attributes) already used by other packages.

To do so, we have used a concept for lua files named "module", inspired by the LaTeX packages. **Warning:** the module concept described here comes as an extention to the base lua module concept. We advise not to use the base lua modules, but the LuaTeXtra module system, as it provides more informations, prints informations in logs, etc. The LuaTeXtra lua module system works basically like the LaTeX package system: you can simply use them, or require them with a version number, etc.

As explained above, another important function provided by LuaTeXtra is attributes allocation with `\newluaattribute`.

1.7 Naming conventions

In LuaTeXtra we decided to adopt a simple naming convention: all macros start by `lua`. There are though a few exceptions: for instance we renamed the concept of LuaTeX's `attribute` to `luaattribute`, as attribute is already something used by a lot of packages and users. This lead us to name our macros `\newluaattribute`, `\unsetluaattribute`, etc. for backward compatibility. On the same principle, `catcodetable` is replaced by `luacatcodetable`.

2 Basic user point of view

2.1 Plain and LaTeX macros

The most useful macro provided by luatextra is certainly `\luadirect`, which is a wrapper for `\directlua`, but will keep the same signature whatever the LuaTeX version. For example this macro works for LuaTeX versions < 0.36, and will most certainly work for future versions of LuaTeX. It is advised to use this macro to keep the compatibility of your package with all versions of LuaTeX. `\lualate` is also provided as a wrapper for `\latelua`.

Some small but useful macros are `\LuaTeX` and `\LuaLaTeX` that produce LaTeX and LuaLaTeX.

luatextra automatically loads luaotfload, so you can open otf fonts directly, for instance `\font\foo= "LinLibertine.otf"\foo` will work. You can also pass options to the font opening, for example the different OTF options you want.

These options are passed with the XeTeX font syntax (briefly described in `XeTeX-reference.pdf`), for example if you want to open a font with the usual TeX substitutions and ligatures, with also the font default ligatures, you can try
`\font\foo= "LinLibertine.otf;+tlig;+tsub;+liga;+rlig;"\foo`.

2.2 LaTeX-only macros

2.2.1 luainputenc

From the user point of view, adapting an old document for LuaTeX is really easy: replacing inputenc by luainputenc in the preamble is enough.

Warning: the current version of luainputenc automatically loads the LuaTeXtra packages. It won't necessarily be the case in the future. So if you use some functions of LuaTeXtra (like `\newluaattribute`), it is advised to also explicitely use `luatextra`. A typical part of LaTeX preamble for LuaTeX will be for example:

```
\usepackage[utf8]{luainputenc}
\usepackage{luatextra}
```

Another important thing to know, for LaTeX, is that loading luainputenc is necessary if you use old fonts. One could believe that LuaTeX reading UTF-8 natively, inputenc is not necessary anymore. It's not the case! The exception to this is the case where you only use OTF fonts. But you can't simply suppress inputenc for your old documents.

Note that luainputenc automatically loads inputenc if called with an old engine, so you will still be able to compile your documents with pdfTeX without changing them.

A new font encoding has been added: `EU2`. It is meant to be used with OTF fonts. It is very minimal and just defines a default font (lmodern). Use it if you use only OTF fonts. To use it simply call fontenc with the option `EU2`.

EU2 encoding is special as it needs non-ASCII characters to be non-active (unlike other font encodings), so you cannot mix old encodings and EU2. If you want to use both old fonts and OTF fonts in your document, the way to do so is by using a new option of luainputenc: `lutf8x`. This option overrides LaTeX's mechanism to change the font encoding and activates or unactivates non-ASCII characters. If you use this option, you will be able to change the encoding, for example:

```
abc
{
\fontencoding{EU2}\usefont
\font\foo="MyOtfFont.otf"\foo
abc
}
abc
```

Another new option has been added to luainputenc: `unactivate`. With this option, non-ASCII characters won't be activated, so you will be able to use only OTF fonts. Typically you can use this options with the `EU2` fontenc option.

2.2.2 luacode environment

LuaTEXtra also provides an environment for lua code integration: luacode. You can simply use it this way:

```
\begin{luacode}
texio.write_nl("It works!")
\end{luacode}
```

3 Advanced user point of view

3.1 lua modules

For more informations about this part (and the attributes and catcodetables), see the documentation `luatextra.pdf`.

LuaTEXtra provides the same kind of functions for lua modules as LATEX does for packages: it provides the macros `\luaUseModule` and `\luaRequireModule`, as well as the functions `luatextra.use_module`, `luatextra.require_module` and `luatextra.provides_module`.

If you want to use a lua file mypackage.lua, you can safely use `\luaUseModule{mypackage}`: it will automatically load the file; and will load it only once if the function is called several times. You can also use `\luaRequireModule` that takes the minimal version (or date) as a second (optional under LATEX) argument.

In a lua module, you have to call `luatextra.provides_module`. It takes a table as an argument. This table must contain at least the fields `name`, `version`, `date` and `description`. You can also use the fields `author`, `copyright`, `license`, or any field you want.

See section 3.6 for an example of a lua module.

3.2 attributes

We already talked about the most important function: `\newluaattribute`. It is the equivalent of `\newcount` for attributes.

Another important function is `\unsetluaattribute` that unsets an attribute. It's adised to use this macro instead of unsetting it by hand, as the unset value may change (it changed in version 0.37 from `-1` to `-"7FFFFFFF`).

You can also use the macro `\setluaattribute` that takes an attribute and a value as arguments, and set the argument to the asked value.

3.3 catcodetables

The equivalent of `\newcount` for catcodetables is `\newluacatcodetable`. Also a set of predefined catcodetables comes with LuaTEXtra:

- `\CatcodeTableIniTeX`: the base TEX catcodes
- `\CatcodeTableString`: almost all characters have catcode 12

- `\CatcodeTableOther`: all characters have catcode 12 (even space)
- `\CatcodeTableLaTeX`: the LaTeX classical catcodes

3.4 callbacks

For more informations about this part, see the documentation `luamcallbacks.pdf`.

One of the main goals of LuaTeXtra is also to provide a way for several packages to register their functions in one callback. The function `callback.register` does not provide any way to do so. To fix it, LuaTeXtra provides the function `callback.add` that takes the same two first arguments as `callback.register` (name of the callback and function to register), and two other arguments: a mandatory description of the function, and an optional priority (the lower the number, the higher the priority).

If people use `callback.register` all the LuaTeXtra mechanism is broken, so we override `callback.register` with a function that simply outputs an error message.

Three other very useful new functions are provided:

- `callback.remove` takes the name of a callback and the description of a register function (third argument of `callback.add`), and removes the function from a callback.
- `callback.reset` removes all register functions from a callback.
- `callback.get_priority` takes the name of a callback and the description of a registered function, and returns the priority of the function in the callback, or 0 if the function is not registered.

3.5 lua extra functions

Some extra functions have been added to the standard lua library, mostly coming from the ConTeXt libraries, for the complete list of functions, please refer to the documentation `lualibs.pdf`.

The most useful of these added functions is certainly `table.serialize` that returns the string describing a given table. This string can be used for debugging purposes, or table saving in files, as the string can be read by lua to rebuild the original table.

3.6 `.lua` module template

```
mypackage = { }

mypackage.module = {
    name          = "mypackage",
    version       =  1.0,
    date          = "2009/03/08",
    description   = "My fancy package",
```

```
    author        = "Me",
    copyright     = "Myself",
    license       = "My License",
}

luatextra.provides_module(mypackage.module)

local format = string.format

function mypackage.log (...)
    luatextra.module_log('mypackage', format(...))
end

function mypackage.warning (...)
    luatextra.module_warning('mypackage', format(...))
end

function mypackage.error (...)
    luatextra.module_error('mypackage', format(...))
end
```

3.7 .sty template

```
\expandafter\ifx\csname ProvidesPackage\endcsname\relax
  \input luatextra.sty
\else
  \NeedsTeXFormat{LaTeX2e}
  \ProvidesPackage{mypackage}
    [2009/03/08 v1.00 My super package.]
  \RequirePackage{luatextra}
\fi

\luaUseModule{mymodule}
```