A guide to LuaLaTeX

Manuel Pégourié-Gonnard <mpg@elzevir.fr>

May 5, 2013

This document is a map, or touristic guide, for the new world of LuaLaTeX.[1] The intended audience ranges from complete newcomers (with a working knowledge of conventional LaTeX) to package developers. This guide is intended to be comprehensive in the following sense: it contains pointers to all relevant sources, gathers information that is otherwise scattered, and adds introductory material.

Feedback and suggestions for improvement are most welcome. This document is work in progress; thanks for your comprehension and patience.

---

[1] Though focusing on LuaLaTeX, it includes relevant information about LuaTeX with the Plain format, too.

1 Introduction

1.1 Just what is LuaLaTeX?

To answer this question, we need to mention a few details about the TeX world that you may usually ignore: the difference between an *engine* and a *format*. An engine is an actual computer program, while a format is a set of macros executed by an engine, usually preloaded when the engine is invoked with a particular name.

Actually, a format is more or less like a document class or a package, except it is associated with a particular command name. Imagine there is a command `latex-article` that would do the same as `latex`, except you wouldn't need to say `\documentclass{article}` at the beginning of your file. Similarly, in current distributions, the command `pdflatex` is the same as the command `pdftex` except that you don't need to put the instructions to load LaTeX at the beginning of your source file. This is convenient, and slightly more efficient too.

Formats are great because they implement powerful commands using the basic tools an engine provides. However the power of the format is sometimes limited by the engine's tools set, so people started developing more powerful engines so that other people can implement even more powerful formats (or packages). The most famous engines now (except the original TeX) are pdfTeX, XeTeX and LuaTeX.

To further complicate the picture, the original TeX engine produced only DVI files, while its successors may (also) produce PDF file. Each command in your system corresponds to a particular engine with a particular format and a particular output mode. The following table summarizes this: rows are indexed by format, columns by engine, and in each cell the first line is the command for this engine with this format in DVI mode, and the second for the same PDF mode.

	TeX	pdfTeX	XeTeX	LuaTeX
Plain	`tex` (none)	`etex` `pdftex`	(none) `xetex`	`dviluatex` `luatex`
LaTeX	(none) (none)	`latex` `pdflatex`	(none) `xelatex`	`dvilualatex` `lualatex`

We can now answer the title question: LuaLaTeX is the LuaTeX engine with the LaTeX format. Well, this answer isn't very satisfying if you don't know what LuaTeX and LaTeX are.

As you probably know, LaTeX is the general framework in which documents begin with `\documentclass`, packages are loaded with `\usepackage`, fonts are selected in a clever way (so that you can switch to boldface while preserving italics), pages are build with complicated algorithms including support for headers, footers, footnotes, margin notes, floating material, etc. This mostly doesn't change with LuaLaTeX, but new and more powerful packages are available to make parts of the system work in a better way.

So, what's LuaTeX? Short version: the hottest TeX engine right now! Long version: It is the designated successor of pdfTeX and includes all of its core features: direct generation of PDF files with support for advanced PDF features and micro-typographic enhancements to TeX typographic algorithms. The main new features of LuaTeX are:

1. Native support of Unicode, the modern standard for character classification and encoding, supporting all characters in the world, from English to traditional Chinese through Arabic, including a lot of mathematical (or otherwise specialised) symbols.

2. Inclusion of Lua as an embedded scripting language (see section 1.3 for details).

3. A lot of wonderful Lua libraries, including:
    - `fontloader`, supporting modern font formats such as TrueType and OpenType;
    - `font`, allowing advanced manipulation of the fonts from within the document;
    - `mplib`, an embedded version of the graphic program MetaPost;
    - `callback`, providing hooks into parts of the TeX engine that were previously inaccessible to the programmer;
    - utility libraries for manipulating images, pdf files, etc.

Some of these features, such as Unicode support, directly impact all documents, while others merely provide tools that package authors will use to provide you with more powerful commands and other enhancements.

1.2 Switching from LaTeX to LuaLaTeX

As the previous section explains, LuaLaTeX is mostly like LaTeX, with a few differences, and more powerful packages and tools available. Here we present an absolute minimum you should know to produce a document with LuaLaTeX, while the rest of the document provides more details about the available packages.

There are only three differences:

1. Don't load inputenc, just encode your source in UTF-8.

2. Don't load fontenc nor textcomp, but load fontspec.

3. babel works with LuaLaTeX but you can load polyglossia instead.

4. Don't use any package that changes the fonts, but use fontspec's commands instead.

So, you only need to get familiar with fontspec, which is easy: select the main (serif) font with `\setmainfont`, the sans serif font with `\setsansfont` and the mono-spaced (typewriter) font with `\setmonofont`. The argument to these commands is the human-friendly name of the font, for example `Latin Modern Roman` rather than `ec-lmr10`. You probably want to use `\defaultfontfeatures{Ligatures=TeX}` before these commands to keep the usual TeX substitutions (such as `---` for an em-dash) working.

The good news is: you can directly access any font on your operating system (in addition to those of your TeX distribution) including TrueType and OpenType fonts and have access to their most advanced features. It means it is now easy to install for use with LuaLaTeX any modern font you may download or purchase from an editor and benefit from their full potential.

Now for the bad news: it is not always easy to get a list of all available fonts. Under Windows with TEX Live, the command-line tool `fc-list` lists them all, but is not very friendly. Under Mac OS X, the *Fontbook* application lists the fonts of your system, but not those of your TEX distribution. Same with `fc-list` on Linux. More bad news: it is not easy to access your old fonts that way. Happily, more fonts are available in modern formats every day (well, month or year, actually, if you count only good fonts).

En passant, let's mention that the content of this section so far also holds for XƎLATEX, that is, LATEX over XƎTEX. Indeed, XƎTEX shares two of the essential features of LuaTEX: native Unicode and support for modern font formats (but doesn't have the other features of LuaTEX; on the other hand, it is more stable right now). Though their implementations concerning fonts are vastly different, fontspec manages to offer a mostly unified font interface for both XƎLATEX and LuaLATEX.

So, to benefit from the new features of LuaTEX, you must drop a few parts of the old world, namely the fonts that are not available in a modern format (and also the liberty to encode your source how you want, but UTF-8 is so much superior that this one hardly counts). The package luainputenc provides various trade-offs that allow you to regain these parts[2] possibly at the expense of loosing real Unicode support.

That's all you need to know to start producing documents with LuaLATEX. I recommend you have a look at the fontspec manual and actually try to compile a small document using funny fonts. You can then skim over the rest of this document as you wish. Section 5 lists all the other differences between conventional LATEX and LuaLATEX that I'm aware of.

1.3 A Lua-in-TEX primer

Lua is a nice, small language, obviously less surprising and much easier to learn than TEX as a programming language. The essential reference is the very readable book *Programming in Lua*, whose first edition is freely available online. For a quick start, I recommend you read chapters 1 to 5 and have a quick glance at part 3. Note that all the libraries mentioned in chapter 3 are included in LuaTEX, but the `os` library is restricted for security reasons.

Depending on your programming culture, you may be directly interested in the rest of part 1 and part 2 which present more advanced features of the language, but part 4 is useless in a LuaTEX context, unless of course you want to hack LuaTEX itself. Finally, the *Lua reference manual* is available online and comes with a handy index.

Now, let's turn to Lua in LuaTEX. The main way to execute Lua code from the TEX end is the `\directlua` command, which takes arbitrary Lua code as an argument. Conversely, you can pass information from Lua to TEX with `tex.sprint`.[3] For example,

```
the standard approximation $\pi = \directlua{tex.sprint(math.pi)}$
```

prints "the standard approximation $\pi = 3.1415926535898$" in your document. See how easy it is to mix TEX and Lua?

---

[2] While the name suggests it is only about input encodings, the details of LATEX's font encoding implementation imply this package is needed (and works) for old fonts too.

[3] The name probably means "string print" as opposed to "run very fast for a short period of time."

Actually, there are a few gotchas. Let's look at the Lua to TeX way first, it's the simplest (since it's more Lua than TeX). If you look at the LuaTeX manual, you'll see there is another function with a simpler name, `tex.print`, to pass information this way. It works by virtually inserting a full line into your TeX source, whose contents are its argument. In case you didn't know, TeX does many nasty[4] things with full lines of the source: ignoring spaces at the beginning and end of line and appending an end-of-line character. Most of the time, you don't want this to happen, so I recommend using `tex.sprint` which virtually inserts its argument in the current line, so is much more predictable.

If you're enough of a TeXnician to know about catcodes, you'll be happy to know that `tex.print` and its variants give you nearly full control over the catcodes used for tokenizing the argument, since you can specify a catcode table as the first argument. You'll probably want to learn about catcode tables (currently 2.7.6 in the LuaTeX manual) before you're fully happy. If you don't know about catcodes, just skip this paragraph.[5]

Let's look at `\directlua` now. To get an idea about how it works, imagine that it's a `\write` command, but it writes only to a virtual file and immediately arranges for this file to be fed to the Lua interpreter. On the Lua end, the consequence is that each argument of a `\directlua` command has its own scope: local variables from one will not be visible to the other. (Which is rather sane, but always good to know.)

Now, the major gotcha is that before being fed to the Lua interpreter, the argument is first read and tokenised by TeX, then fully expanded and turned back into a plain string. Being read by TeX has several consequences. One of them is that end of lines are turned into spaces, so the Lua interpreter sees only one (long) line of input. Since Lua is a free-form language, it doesn't usually matter, but it does if you use comments:

```
\directlua{a_function()
  -- a comment
  another_function()}
```

won't do what you probably expect: `another_function()` will be seen as part of the comment (it's only one line, remember).

Another consequence of being read by TeX is that successive spaces are merged into one space, and TeX comments are discarded. So, here is a surprisingly correct version of the previous example.

```
\directlua{a_function()
  % a comment
  another_function()}
```

It is also worth noticing that, since the argument basically is inside a `\write`, it's in expansion-only context. If you don't know what it means, let me say that expansion issues are mostly what makes TeX programming so difficult to avoid expanding further on that matter.

---

[4] Okay, these are usually nice and helpful actions, but in this case they are most probably unexpected so I call them nasty.

[5] Erf, too late, you already read it.

I'm sorry if the last three paragraphs were a bit TEXnical in nature but I thought you had to know. To reward you for staying with me, here is a debugging trick. Put the following code near the beginning of your document:

```
\newwrite\luadebug
\immediate\openout\luadebug luadebug.lua
\AtEndDocument{\immediate\closeout\luadebug}
\newcommand\directluadebug{\immediate\write\luadebug}
```

Then, when you have a hard time understanding why a particular call to `\directlua` doesn't do what you expect, replace this instance of the command with `\directluadebug`, compile as usual and look in the file `luadebug.lua` produced what the Lua interpreter actually read.

The luacode package provides commands and environments that help to varying degrees with some of these problems. Hoewever, for everything but trivial pieces of Lua code, it is wiser to use an external file containing only Lua code defining functions, then load it and use its functions. For example:

```
\directlua{dofile("my-lua-functions.lua")}
\newcommand*{\greatmacro}[2]{%
  \directlua{my_great_function("\luatexluaescapestring{#1}", #2)}}
```

The example assumes that `my_great_function` is defined in `my-lua-functions.lua` and takes a string and a number as arguments. Notice that we carefully use the `\luatexluaescapestring` primitives on the string argument to escape any backslash or double-quote it might contain and which would confuse the Lua parser.[6]

That's all concerning Lua in TEX. If you're wondering why `\luatexluaescapestring` has such a long and silly name, you might want to read the next section.

1.4 Other things you should know

Just in case it isn't obvious, the LuaTEX manual, `luatexref-t.pdf`, is a great source of information about LuaTEX and you'll probably want to consult it at some point (though it is a bit arid and technical).

It is important to know that the name of the new primitives of LuaTEX as you read them in the manual are not the actual names you'll be able to use in LuaLATEX. To prevent clashes with existing macro names, all new primitives have been prefixed with `\luatex` unless they already start with it, so `\luaescapestring` becomes `\luatexluaescapetring` while `\luatexversion` remains `\luatexversion`. The rationale is detailed in section 4.

Oh, and by the way, did I mention that LuaTEX is in beta and version 1.0 is expected in spring 2014? You can learn more on the roadmap page of the LuaTEX site. Stable betas are released regularly and are included in TEX Live since 2008 and MikTEX since 2.9.

---

[6] If you ever used SQL then the concept of escaping strings is hopefully not new to you.

Not surprisingly, support for LuaTeX in LaTeX is shiny new, which means it may be full of (shiny) bugs, and things may change at any point. You might want to keep your TeX distribution very up-to-date[7] and also avoid using LuaLaTeX for critical documents at least for some time.

As a general rule, this guide documents things as they are at the time it is written or updated, without keeping track of changes. Hopefully, you'll update your distribution as a whole so that you always get matching versions of this guide and the packages, formats and engine it describes.

2 Essential packages and practices

This section presents the packages you'll probably want to always load as a user, or that you should absolutely know about as a developer.

2.1 User-level

fontspec **Engines:** XeTeX, LuaTeX. **Formats:** LaTeX.
Authors: Will Robertson.
CTAN location: `macros/latex/contrib/fontspec/`.
Development url: `https://github.com/wspr/fontspec/`.
Nice interface to font management, well-integrated in to the LaTeX font selection scheme. Already presented in the previous section.

polyglossia **Engines:** XeTeX, LuaTeX. **Formats:** LaTeX.
Authors: François Charette & Arthur Reuntenauer.
CTAN location: `macros/latex/contrib/polyglossia/`.
Development url: `https://github.com/reutenauer/polyglossia/`.
A simple and modern replacement for Babel, working in synergy with fontspec.

2.2 Developer-level

2.2.1 Naming conventions

On the TeX end, control sequences starting with `\luatex` are reserved for primitives. It is strongly recommended that you do *not* define any such control sequence, to prevent name clashes with future versions of LuaTeX. If you want to a emphasize that a macro is specific to LuaTeX, we recommend that you use the `\lua` prefix (without a following `tex`) instead. It is okay to use the `\luatex@` prefix for internal macros, since primitive names never contain `@`, but it might be confusing. Moreover, you're already using a unique prefix for internal macros in all of your packages, aren't you?

On the Lua end, please keep the global namespace as clean as possible. That is, use a table `mypackage` and put all your public functions and objects in this table. You might want to avoid using Lua's deprecated `module()`. Other strategies for Lua module management are discussed

---

[7] For TeX Live, consider using the complementary tlcontrib repository.

in chapter 15 of *Programming in Lua*, and examples are given in `luatexbase-modutils.pdf`. Also, it is a good and necessary practice to use `local` for your internal variables and functions. Finally, to avoid clashes with future versions of LuaTeX, it is necessary to avoid modifying the namespaces of LuaTeX's default libraries.

2.2.2 Engine and mode detection

Various packages allow to detect the engine currently processing the document.

ifluatex **Engines:** all. **Formats:** LaTeX, Plain.
Authors: Heiko Oberdiek.
CTAN location: `macros/latex/contrib/oberdiek/`.
Provides `\ifluatex` and makes sure `\luatexversion` is available.

iftex **Engines:** all. **Formats:** LaTeX, Plain.
Authors: Vafa Khalighi.
CTAN location: `macros/latex/contrib/iftex/`.
Development url: `http://bitbucket.org/vafa/iftex`.
Provides `\ifPDFTeX`, `\ifXeTeX`, `\ifLuaTeX` and corresponding `\Require` commands.

expl3 **Engines:** all. **Formats:** LaTeX.
Authors: The LaTeX3 Project.
CTAN location: `macros/latex/contrib/expl3/`.
Development url: `http://www.latex-project.org/code.html`.
Amongst *many* other things, provides `\luatex_if_engine:TF`, `\xetex_if_engine:TF` and their variants.

ifpdf **Engines:** all. **Formats:** LaTeX, Plain.
Authors: Heiko Oberdiek.
CTAN location: `macros/latex/contrib/oberdiek/`.
Provides `\ifpdf` switch. LuaTeX, like pdfTeX, can produce either PDF or DVI output; the later is not very useful with LuaTeX as it doesn't support any advanced feature such as Unicode and modern font formats. The `\ifpdf` switch is true if and only if you are running pdfTeX-or-LuaTeX in PDF mode (note that this doesn't include XeTeX, whose support for PDF is different).

2.2.3 Basic resources

luatexbase **Engines:** LuaTeX. **Formats:** LaTeX, Plain.
Authors: Élie Roux, Manuel Pégourié-Gonnard & Philipp Gesang.
CTAN location: `macros/luatex/generic/luatexbase/`.
Development url: `https://github.com/lualatex/luatexbase`.
The Plain and LaTeX formats provide macros to manage TeX basic resources, such as count or box registers. LuaTeX introduces new resources that need to be shared gracefully by packages. This package provides the basic tools to manage: the extended conventional TeX resources, catcode tables, attributes, callbacks, Lua modules loading and identification. It also provides basic tools to handle a few compatibility issues with older version of LuaTeX.

Warning. This package is currently in conflict with the luatex package, since they both do almost the same thing. The respective package authors are well aware of this situation and plan to somehow merge the two packages in the near future, though the timeline is not clear.

luatex **Engines:** LuaTeX. **Formats:** LaTeX, Plain.
Authors: Heiko Oberdiek.
CTAN location: `macros/latex/contrib/oberdiek/`.
See the description of luatexbase above. This package provides the same core features except for callback management and Lua module identification.

lualibs **Engines:** LuaTeX. **Formats:** Lua.
Authors: Élie Roux & Philipp Gesang.
CTAN location: `macros/luatex/generic/lualibs/`.
Development url: `https://github.com/lualatex/lualibs`.
Collection of Lua libraries and additions to the standard libraries; mostly derived from the ConTeXt libraries. If you need a basic function that Lua doesn't provide, check this package before rolling your own implementation.

2.2.4 Font internals

Those packages are loaded by fontspec to handle some low-level font and encoding issues. A normal user should only use fontspec, but a developer may need to know about them.

luaotfload **Engines:** LuaTeX. **Formats:** LaTeX, Plain.
Authors: Élie Roux, Khaled Hosny & Philipp Gesang.
CTAN location: `macros/luatex/generic/luaotfload/`.
Development url: `https://github.com/lualatex/luaotfload`.
Low-level OpenType font loading, adapted from the generic subset of ConTeXt. Basically, it uses the `fontloader` Lua library and the appropriate callbacks to implement a syntax for the `\font` primitive very similar to that of XeTeX and implement the corresponding font features. It also handles a font database for transparent access to fonts from the system and the TeX distribution either by family name or by file name, as well as a font cache for faster loading.

euenc **Engines:** XeTeX, LuaTeX. **Formats:** LaTeX.
Authors: Will Robertson, Élie Roux & Khaled Hosny.
CTAN location: `macros/latex/contrib/euenc/`.
Development url: `https://github.com/wspr/euenc`.
Implements the EUx Unicode font encodings for LaTeX's fontenc system. Currently, XeLaTeX is using `EU1` and LuaLaTeX is using `EU2`. Includes definitions (`fd` files) for Latin Modern, the default font loaded by fontspec.

To be precise, euenc merely declares the encoding, but doesn't provide definitions for LICR macros; this is done by loading xunicode with `\UTFencname` defined to `EU1` or `EU2`, which fontspec does. The actual encodings are the same, but it is useful to have distinct names so that different `fd` files can be used according to the engine (which is actually the case with Latin Modern).

3 Other packages

Note that the packages are not listed in any particular order.

3.1 User-level

luatextra **Engines:** LuaTeX. **Formats:** LaTeX.
Authors: Élie Roux & Manuel Pégourié-Gonnard.
CTAN location: `macros/luatex/latex/luatextra/`.
Development url: `https://github.com/lualatex/luatextra`.
Loads usual packages, currently fontspec, luacode, metalogo (commands for logos, including `\LuaTeX` and `\LuaLaTeX`), luatexbase, lualibs, fixltx2e (fixes and enhancements for the LaTeX core).

luacode **Engines:** LuaTeX. **Formats:** LaTeX.
Authors: Manuel Pégourié-Gonnard.
CTAN location: `macros/luatex/latex/luacode/`.
Development url: `https://github.com/lualatex/luacode`.
Provides commands and macros that help including Lua code in a TeX source, especially concerning special characters.

luainputenc **Engines:** LuaTeX, XeTeX, pdfTeX. **Formats:** LaTeX.
Authors: Élie Roux & Manuel Pégourié-Gonnard.
CTAN location: `macros/luatex/latex/luainputenc/`.
Development url: `https://github.com/lualatex/luainputenc`.
Helps compiling documents relying on legacy encodings (either in the source or with the fonts). Already presented in the introduction. When running XeTeX, just loads xetex-inputenc; under pdfTeX, loads the standard inputenc.

luamplib **Engines:** LuaTeX. **Formats:** LaTeX, Plain.
Authors: Hans Hagen, Taco Hoewater & Philipp Gesang.
CTAN location: `macros/luatex/generic/luamplib/`.
Development url: `https://github.com/lualatex/luamplib`.
Provides a nice interface for the `mplib` Lua library that embeds metapost in LuaTeX.

luacolor **Engines:** LuaTeX. **Formats:** LaTeX.
Authors: Heiko Oberdiek.
CTAN location: `macros/latex/contrib/oberdiek/`.
Changes low-level color implementation to use LuaTeX attributes in place of whatsits. This makes the implementation more robust and fixes strange bugs such as wrong alignment when `\color` happens at the beginning of a `\vbox`.

3.2 Developer-level

pdftexcmds **Engines:** LuaTeX, pdfTeX, XeTeX. **Formats:** LaTeX, Plain.
Authors: Heiko Oberdiek.

CTAN location: `macros/latex/contrib/oberdiek/`.
Though LuaTEX is mostly a superset of pdfTEX, a few utility primitives were removed (those that are sort of superseded by Lua) or renamed. This package provides them with consistent names across engines, including XeTEX which recently implemented some of these primitives, such as `\strcmp`.

magicnum **Engines:** LuaTEX, pdfTEX, XeTEX. **Formats:** LaTEX, Plain.
Authors: Heiko Oberdiek.
CTAN location: `macros/latex/contrib/oberdiek/`.
Provides hierarchical access to ‘‘magic numbers’’ such as catcodes, group types, etc. used internally by TEX and its successors. Under LuaTEX, a more efficient implementation is used and a Lua interface is provided.

lua-alt-getopt **Engines:** `texlua`. **Formats:** command-line.
Authors: Aleksey Cheusov.
CTAN location: `support/lua/lua-alt-getopt`.
Development url: `https://github.com/LuaDist/alt-getopt`.
Command-line option parser, mostly compatible with POSIX and GNU getopt, to be used in command-line Lua scripts such as `mkluatexfontdb` from luaotfload.

4 The `luatex` and `lualatex` formats

This section is for developers and curious users only; normal users can safely skip it. The following information apply to TEX Live 2010, and most likely to MikTEX 2.9 too, though I didn't actually check. Earlier versions of TEX Live had slightly different and less complete arrangements.

Primitive names As mentioned in section 1.4, the names of the LuaTEX-specific primitives are not the same in the `lualatex` format as in the LuaTEX manual. In the `luatex` format (that is, LuaTEX with the Plain format), primitives are available with their natural name, but also with the prefixed name, in order to ease development of generic packages.

The rationale, copy-pasted from the file `lualatexiniconfig.tex` that implements this for the `lualatex` format, is:

1. All current macro packages run smoothly on top of pdf(e)TeX, so those primitives are left untouched.
2. Other non-TeX82 primitives in LuaTEX may cause name clashes with existing macros in macro packages, especially when they use very ‘‘natural’’ names such as `\outputbox`, `\mathstyle` etc. Such a probability for name clashes is undesirable, since the most existing LaTeX documents that run without change under LuaTEX, the better.
3. The LuaTEX team doesn't want to apply a systematic prefixing policy, but kindly provided a tool allowing prefixes to be applied. So we chose to use it. Previously, we even disabled the extra primitives, but now we feel it's better to enable them with systematic prefixing, in order to avoid that each and every macro package (or user) enables them with various and inconsistent prefixes (including the empty prefix).

4. The `luatex` prefix was chosen since it is already used as a prefix for some primitives, such as `\luatexversion`: this way, those primitives don't end up with a double prefix (for details, see `tex.enableprimitives` in the LuaTeX manual).
5. The `\directlua` primitive is provided both with its natural name (allowing easy detection of LuaTeX) and a prefixed version `\luatexdirectlua` (for consistency with `\luatexlatelua`).
6. Various remarks:
   - The obvious drawback of such a prefixing policy is that the names used by LaTeX or generic macro writer won't match the names used in the manual. We hope this is compensated by the gain in backwards compatibility.
   - All primitives dealing with Unicode math already begin with `\U`, and maybe will match the names of XeTeX primitives some day, so maybe prefixing was not necessary/desirable for them. However, we tried to make the prefixing rule as simple as possible, so that the previous point doesn't get even worse.
   - Maybe some day we'll feel it's better to provide all primitives without prefixing. If this happens, it will be easy to add the unprefixed primitives in the format while keeping the prefixed names for compatibility. It wouldn't work the other way round; i.e., belatedly realizing that we should not provide the unprefixed primitives would then break any LuaTeX-specific macro packages that had been written.

`\jobname` The LaTeX kernel (and a lot of packages) use constructs like `\input\jobname.aux` for various purposes. When `\jobname` contains spaces, this doesn't do the right thing, since the argument of `\input` ends at the first space. To work around this, pdfTeX automagically quotes `\jobname` when needed, but LuaTeX doesn't for some reason. A nearly complete workaround is included in LaTeX-based (as opposed to Plain-based) LuaTeX formats.

It doesn't work, however, if LuaTeX is invoked as `lualatex '\input name'`, as opposed to the more usual `lualatex name`. To work around this limitation of the workaround included in the format, specifying a jobname explicitly, as in `lualatex jobname=name '\input name'`. Or even better, just don't use spaces in the names of your TeX files.

For more details, see this old thread and this newer one on the LuaTeX mailing lists, and `lualatexquotejobname.tex` for the implementation of the workaround.

hyphenation LuaTeX offers dynamic loading for hyphenation patterns. The support for this in babel and polyglossia appeared only on TeX Live 2013, but should work well since then.

Documentation and implementation details are included in `luatex-hyphen.pdf`. The sources are part of the texhyphen project.

codes The engine itself does not set `\catcode`s, `\lccode`s, etc. for non-ASCII characters. Correct `\lccode`s in particular are essential for hyphenation to work. Formats for LuaTeX now include `luatex-unicode-letters.tex`, a modified version of `unicode-letters.tex` from the XeTeX distribution, that takes care of settings these values in accordance with the Unicode standard.

This was added after TeX Live 2010 went out, so you are strongly advised to update your installation if you want to enjoy proper hyphenation for non-ASCII text.

5 Things that just work, partially work, or don't work (yet)

5.1 Just working

Unicode Conventional LaTeX offers some level of support for UTF-8 in input files. However, at a low level, non-ASCII characters are not atomic in this case: they consist of several elementary pieces (known as *tokens* to TeXnicians). Hence, some packages that scan text character by character or do other atomic operations on characters (such as changing their catcodes) often have problems with UTF-8 in conventional LaTeX. For example, you can't use any non-ASCII character for short verbatim with fancyvrb, etc.

The good news is, with LuaLaTeX, some of these package's features start working on arbitrary Unicode characters without needing to modify the package. The bad news is, this isn't always true. See the next section for details.

5.2 Partially working

microtype Package microtype has limited support for LuaTeX: more precisely, as of version 2.5 2013/03/13, protrusion and expansion are available and activated by default in PDF mode, but kerning, spacing and tracking are not supported (see table 1 in section 3.1 of `microtype.pdf`).

On the other hand, luaotfload, loaded by fontspec, supports a lot of microtypographic features. So the only problem is the lack of a unified interface.

xunicode Package xunicode's main feature is to ensure that the usual control sequences for non-ASCII characters (such as `\'e`) do the right thing in a Unicode context. It could *probably* work with LuaTeX, but explicitly checks for XeTeX only. However, fontspec uses a trick to load it anyway. So, you can't load it explicitly, but you don't need to, since fontspec already took care of it.

encodings As mentioned in the above section, a few things that were problematic with UTF-8 on conventional LaTeX spontaneously works, but not always. For example, with the listings package on LuaLaTeX, you may use only characters below 256 (that is, characters from the Latin-1 set), inside your listings (but of course the full Unicode range is still available in the rest of your document).

metrics This item isn't exactly about working or not working, but rather about not working in exactly the same way as pdfTeX or XeTeX: you may observe minor differences in the layout and hyphenation of your text.

They may be due variations between two versions of the same font used by the various engines, adjustments made to the hyphenation, ligaturing or kerning algorithms (or example, the first word of a paragraph, as well as words containing different fonts, can now be hyphenated), or differences in the hyphenation patterns used (patterns used by pdfTeX are basically frozen, but LuaTeX and XeTeX use newer version for some languages) for this language.

If you ever observe a major difference between pdfLaTeX and LuaLaTeX with the same fonts, it is not at all unlikely that a bug in LuaTeX[8] or in the font is involved. As usual, make sure your distribution is up-to-date before reporting such a problem.

---

[8] For example, LuaTeX 0.60 had a bug that prevented any hyphenation after a `---` ligature until the end of the paragraph.

babel Mostly working mostly without problems for Latin languages. For other languages, your mileage may vary. Even for Latin languages, encoding-related problems my happen.

polyglossia A more modern, but less complete, package for multilingual support, polyglossia, is also available and should be preffered, though it does not support all babel features yet.

5.3 Not working (yet)

old encodings Packages playing with input (source files) or output (fonts) encodings are very likely to break with LuaTeX. This includes inputenc, fontenc, textcomp, and probably most classical font packages such as mathptmx or fourier. The good news is, Unicode is a more powerful way to handle encoding problems that old packages were trying to solve, so you most likely don't need these packages anyway. However, not everything is already ported to the shiny new world of Unicode, so you may have a more limited (or just different) set of choices available for some time (this is especially true for fonts).

spaces Spaces in file names are not really well supported in the TeX world in general. This doesn't really get better with LuaTeX. Also, due to tricky reasons, things may be worse if you have spaces in the name of your main TeX file *and* don't invoke LuaTeX in the usual way. If you do invoke it in the usual way, everything should work, and I won't tell you what the unusual invocation looks like. Otherwise, read the point about jobname in section 4 for a workaround and technical details. Or even better, don't use spaces in the names of your TeX files.