LuaTEX Reference

Snapshot 2006–10–13

Contents

1 Introduction

This book will eventually become the reference manual of LUAT_EX. At the moment, it simply reports the behavior of the executable matching the snapshot date in the title page.

Features may come and go. The current version of LUAT_EX is not meant for production and users cannot depend on functionality staying the same.

Nothing in the API is considered stable just yet. This manual therefore simply reflects the current state of the executable. **Absolutely nothing** on the following pages is set in stone. When the need arises, anything can (and will) be changed without prior notice.

If you are unhappy with this situation, wait for the public beta's.

LUAT_EX consists of a number of interrelated but (still) distinguishable parts:

- PDFT_EX version 1.40 (currently in beta)
- ALEPH RC4 (from the T_EXLIVE repository)
- Functionality of ε-T_EX 2.2
- Lua 5.1
- Dedicated lua libraries
- Various T_EX extensions
- Compiled source code to glue it all together

LUAT_EX has two separate identities:

1. When `\pdfoutput` is set to one, LUAT_EX behaves like PDFT_EX, with the addition of (8-bit) OTP processing. In this mode, fonts are limited to 256 characters, and hyphenation is only available for 8-bit font encodings. Attempts to use the ALEPH direction commands will generate erroneous output.
2. When `\pdfoutput` is zero, LUAT_EX behaves like ALEPH with the addition of the micro-typography features. In this mode, fonts can have 65536 characters, and the whole Unicode base plane can be hyphenated (assuming a proper font encoding). The PDFT_EX commands that are not specific to the PDF output format should work.

In either mode, I/O translation processes, tcx files, enctex, cannot be used. The encoding items are superseded by a LUA-based solution (`reader` callbacks).

2 Basic TEX enhancements

2.1 Unicode support

Text input and output is now considered to be Unicode text, so characters can use the full range of Unicode ($2^{20} + 2^{16} = $ "10FFFF = 1114111).

For now, it only makes sense to use values above the base plane ("FFFF) for `\mathcode` and `\catcode` assignments, since the fonts as well as the hyphenation patterns are still limited to at the most 16–bit values, so the other command will not know what to do with those high values.

Many primitives are affected by this. For instance, `\char` now accepts values between 0 and 1114111. This should not be a problem for well–behaved input files, but it could create incompatibilities for input that would have generated an error when processed by older TEX–based engines.

Primitive	Bits	Hex	Range
`\char`	21	"10FFFF	$(2^{20} + 2^{16})$
`\chardef`	21=21	"10FFFF="10FFFF	$(2^{20} + 2^{16}) = (2^{20} + 2^{16})$
`\lccode`	21=21	"10FFFF="10FFFF	$(2^{20} + 2^{16}) = (2^{20} + 2^{16})$
`\uccode`	21=21	"10FFFF="10FFFF	$(2^{20} + 2^{16}) = (2^{20} + 2^{16})$
`\sfcode`	21=15	"10FFFF="7FFF	$(2^{20} + 2^{16}) = (2^{15})$
`\catcode`	21=4	"10FFFF="F	$(2^{20} + 2^{16}) = (2^{4})$
`\mathchardef`	21=15	"10FFFF="8000	$(2^{20} + 2^{16}) = (2^{3} * 2^{8} * 2^{4})$
`\mathcode`	21=15	"10FFFF="8000	$(2^{20} + 2^{16}) = (2^{3} * 2^{8} * 2^{4})$
`\delcode`	21=27	"10FFFF="7FFFFFF	$(2^{20} + 2^{16}) = (2^{3} * 2^{4} * 2^{8} * 2^{4} * 2^{8})$

As far as the core engine is aware, all input and output to text files is UTF–8 encoded. Input files can be preprocessed using the `reader` callback. This will be explained in a later chapter.

Output to the terminal uses `^^` notation for the lower control range, with the exception of `^^I`, `^^J` and `^^M`. These are considered 'safe' and therefore printed as–is.

No normalization of the Unicode input takes place yet, but that will happen eventually.

2.2 Wide math characters

Text is now extended up to the full Unicode range, but math mode deals mostly with glyphs in fonts directly, and fonts tend to be 16–bit at maximum.

Therefore, the math primitives from ALEPH are kept mostly as–is, except for the ones that convert from input to math commands. The extended commands (with the 'o' prefix) accept 16–bit glyph indices in one of 256 possible families. The traditional TEX primitives are unchanged, their arguments are up–scaled internally.

Primitive	Bits	Hex	Range
\mathchar	15	"7FFF	$(2^3 * 2^8 * 2^4)$
\delimiter	27	"7FFFFFF	$(2^3 * 2^4 * 2^8 * 2^4 * 2^8)$
\omathchar	27	"7FFFFFF	$(2^3 * 2^{16} * 2^8)$
\odelimiter	27+24	"7FFFFFF+"FFFFFF	$(2^3 * 2^8 * 2^{16}) + (2^8 * 2^{16})$
\omathchardef	21=27	"10FFFF="8000000	$(2^{20} + 2^{16}) = (2^3 * 2^{16} * 2^8)$
\omathcode	21=27	"10FFFF="8000000	$(2^{20} + 2^{16}) = (2^3 * 2^{16} * 2^8)$
\odelcode	21=27+24	"10FFFF="7FFFFFF+ "FFFFFF	$(2^{20} + 2^{16}) = (2^3 * 2^8 * 2^{16}) + (2^8 * 2^{16})$

2.3 Extended register tables

All registers can be `<16-bit number>`, as in ALEPH. The affected commands are:

\count	\unhbox
\dimen	\unvbox
\skip	\copy
\muskip	\unhcopy
\marks	\unvcopy
\toks	\wd
\countdef	\ht
\dimendef	\dp
\skipdef	\setbox
\muskipdef	\vsplit
\toksdef	
\box	

2.4 Lua related primitives

In order to merge lua code with TEX input, a few new primitives are needed. LUATEX has support for 65536 separate lua interpreter states. States are automatically created based on the integer argument to the primitives \directlua and \latelua.

2.4.1 \directlua

The primitive \directlua is used to execute lua code. The syntax is

\directlua ⟨16-bit number⟩ ⟨general text⟩

The ⟨general text⟩ is fed into the lua interpreter state indicated by the ⟨16-bit number⟩. If the state does not exist yet, then it will be initialized automatically.

This command is expandable.

2.4.2 \latelua

\latelua stores lua code in a whatsit that will be processed inside the output routine. It is very similar to \pdfliteral.

Any output it produces should go straight to the PDF file, but at the moment it is broken (there is no way to generate the desired output)

  \latelua ["direct"|"page"] ⟨16-bit number⟩ ⟨general text⟩

2.4.3 \luaescapestring

This primitive converts a TEX token string so that it can be safely used as the contents of a Lua string: embedded backslashes, double quotes and single quotes are escaped by prepending an extra token consisting of a backslash with catcode 12.

  \luaescapestring ⟨general text⟩

2.4.4 \luaclose

This primitive allows you to close a lua state, freeing all of its used memory.

  \luaclose ⟨16-bit number⟩

You cannot close lua state zero (0), any attempt to do so will be silently ignored.

States are only closed automatically when a fatal (out of memory) error occurs, but at that point LUATEX will exit anyway.

2.5 New ε-TEX primitives

2.5.1 \clearmarks

This primitive clears a marks class completely, resetting all three connected mark texts to empty.

  \clearmarks ⟨16-bit number⟩

2.5.2 \formatname

\formatname's syntax is identical to \jobname.

In initex, the expansion is empty. Otherwise, the expansion is the value that \jobname had during the initex run that dumped the currently loaded format.

2.5.3 \scantextokens

The syntax of \scantextokens is identical to \scantokens.

This is a slightly adapted version of ε-T$_E$X's **\scantokens**. The differences are:

- The last (and usually only) line does not have a **\endlinechar** appended
- **\scantextokens** never raises an EOF error, and it does not execute **\everyeof** tokens.
- The 'while end of file' tests are not executed, allowing the expansion to end on a different grouping level or while a conditional is still incomplete

2.5.4 Catcode tables

Catcode tables are a new feature that allows you to switch to a predefined catcode regime in a single statement. You can have a practically unlimited number of different tables (at this moment up to 268,435,456. The limit depends on an array allocation).

The subsystem is backward compatible: if you never use the following commands, your document will not notice any difference in behavior compared to traditional T$_E$X.

The contents of each catcode table is independent of any other catcode tables, and their contents is stored and retrieved from the format file.

2.5.4.1 \catcodetable

 \catcodetable ⟨28-bit number⟩

The **\catcodetable** switches to a different catcode table. Such a table has to be previously created using one of the two primitives below, or it has to be zero (table zero is initialized by initex)

2.5.4.2 \initcatcodetable

 \initcatcodetable ⟨28-bit number⟩

The **\initcatcodetable** creates a new table with catcodes identical to those defined by initex:

^^M (<return>)	car_ret	5
(space)	spacer	10
\\	escape	0
%	comment	14
^^? (<delete>)	invalid_char	15
^^@ (<null>)	ignore	9
a–z	letter	11
A–Z	letter	11
everything else	other	12

The new catcode table is allocated globally: it will not go away after the current group has ended. If the supplied number is the currently active table, an error is raised.

2.5.4.3 \savecatcodetable

\savecatcodetable ⟨28-bit number⟩

\savecatcodetable copies the current set of catcodes to a new table with the requested number. The definitions in this new table are all treated as if they were made in the outermost level.

The new table is allocated globally: it will not go away after the current group has ended. If the supplied number is the currently active table, an error is raised.

3 Lua general

3.1 Initialization

Whenever the LUATEX executable starts, it looks for a `--lua` command—line option. If such an option is present, it will enter an alternative mode of command—line parsing.

In this mode, it will only interpret a very small subset of the command—line directly:

–lua=s   load and execute a lua init script
–help   display help and exit
–version   display version and exit

If the Lua script can not be found using the actual name of the command—line, a second attempt is made by prepending the value of the environment variable `LUATEXDIR`, if it esists.

Then the script is loaded and executed. It will find the entire commandline in the table `arg`, beginning with `arg[0]`, that is the name of the executable. At the end of script execution, LUATEX will fetch some of the processed command line options from the `texconfig` table (see the description of the `texconfig` table later on in this document).

This happens very early on. So early, in fact, that none of TEX's initializations have taken place yet. For that reason, the `tex` and `pdf` tables are off-limits during the execution of the startup file (they are nilled). Special care is taken that `texio.write` and `texio.write_nl` function properly, so that you can at least report your actions to the log file when it eventually becomes opened (note that TEX does not even know it's `\jobname` yet at this point).

The file is loaded into Lua state 0, and everything you do will remain visible during the rest of the run, with the exception of the `tex` and `pdf` tables: those will be restored to their normal meaning right after the execution of the script.

We recommend you use the startup file only for your own TEX-independant initializations (if you need any), to parse the command—line, set values in the `texconfig` table, and register the callbacks you need.

Unless the `texconfig` table tells it not to start kpathsea at all (set `texconfig.kpse_init` to `false` for that), it also acts on three other command—line options:

–fmt=s   set the format name
–progname=s   set the progname (only for kpathsea)
–ini   enable initex mode

In order to initialize the built-in kpathsea library properly, LUATEX needs to know the correct 'prog-name' to use, and for that it needs to check `-progname` and `-ini` and `-fmt` (if `-progname` is missing).

3.2 Lua changes

Two modules that are normally external are statically linked in with LUATEX: `slnunicode` and `luazip`.

The `read("*line")` function from the io library has been adjusted so that it is line-ending neutral: any of `LF`, `CR` or typeCR+LF are accepted.

4 Lua Libraries

The interfacing between TEX and Lua is facilitated by a set of Lua modules.

4.1 The tex library

The tex table contains a large list of virtual internal TEX parameters that are partially writable.

The designation 'virtual' means that these items are not properly defined in Lua, but are only fron–tends that are handled by a metatable that operates on the actual TEX values. As a result, most of the lua table operators (like `pairs` and `#`) do not work on such items.

At the moment, it is possible to access almost every parameter that has these characteristics:

- You can use it after `\the`
- It is a single token.

This excludes parameters that need extra arguments, like `\the\scriptfont`.

The subset comprising simple integer and dimension registers are writable as well as readable (stuff like `\tracingcommands` and `\parindent`).

4.1.1 Integer parameters

The integer parameters accept and return lua numbers.

Read–write:

`tex.adjdemerits`	`tex.globaldefs`
`tex.binoppenalty`	`tex.hangafter`
`tex.brokenpenalty`	`tex.hbadness`
`tex.catcodetable`	`tex.holdinginserts`
`tex.clubpenalty`	`tex.hyphenpenalty`
`tex.day`	`tex.interlinepenalty`
`tex.defaulthyphenchar`	`tex.language`
`tex.defaultskewchar`	`tex.lastlinefit`
`tex.delimiterfactor`	`tex.lefthyphenmin`
`tex.displaywidowpenalty`	`tex.linepenalty`
`tex.doublehyphendemerits`	`tex.localbrokenpenalty`
`tex.endlinechar`	`tex.localinterlinepenalty`
`tex.errorcontextlines`	`tex.looseness`
`tex.escapechar`	`tex.mag`
`tex.exhyphenpenalty`	`tex.maxdeadcycles`
`tex.fam`	`tex.month`
`tex.finalhyphendemerits`	`tex.newlinechar`
`tex.floatingpenalty`	`tex.outputpenalty`

```
tex.pausing                          tex.predisplaypenalty
tex.pdfadjustinterwordglue           tex.pretolerance
tex.pdfadjustspacing                 tex.relpenalty
tex.pdfappendkern                    tex.righthyphenmin
tex.pdfcompresslevel                 tex.savinghyphcodes
tex.pdfdecimaldigits                 tex.savingvdiscards
tex.pdfforcepagebox                  tex.showboxbreadth
tex.pdfgamma                         tex.showboxdepth
tex.pdfgentounicode                  tex.time
tex.pdfimageapplygamma               tex.tolerance
tex.pdfimagegamma                    tex.tracingassigns
tex.pdfimagehicolor                  tex.tracingcommands
tex.pdfimageresolution               tex.tracinggroups
tex.pdfinclusionerrorlevel           tex.tracingifs
tex.pdfminorversion                  tex.tracinglostchars
tex.pdfmovechars                     tex.tracingmacros
tex.pdfobjcompresslevel              tex.tracingnesting
tex.pdfoptionalwaysusepdfpagebox     tex.tracingonline
tex.pdfoptionpdfinclusionerrorlevel  tex.tracingoutput
tex.pdfoptionpdfminorversion         tex.tracingpages
tex.pdfoutput                        tex.tracingparagraphs
tex.pdfpagebox                       tex.tracingrestores
tex.pdfpkresolution                  tex.tracingscantokens
tex.pdfprependkern                   tex.tracingstats
tex.pdfprotrudechars                 tex.uchyph
tex.pdftracingfonts                  tex.vbadness
tex.pdfuniqueresname                 tex.widowpenalty
tex.postdisplaypenalty               tex.year
tex.predisplaydirection
```

Read–only:

`tex.deadcycles`	`tex.prevgraf`
`tex.insertpenalties`	`tex.spacefactor`
`tex.parshape`	

4.1.2 Dimension parameters

The dimension parameters accept lua numbers (signifying scaled points) or strings (with included dimension). The result is always a string.

Read–write:

`tex.boxmaxdepth`	`tex.pdfdestmargin`
`tex.delimitershortfall`	`tex.pdfeachlinedepth`
`tex.displayindent`	`tex.pdfeachlineheight`
`tex.displaywidth`	`tex.pdffirstlineheight`
`tex.emergencystretch`	`tex.pdfhorigin`
`tex.hangindent`	`tex.pdflastlinedepth`
`tex.hfuzz`	`tex.pdflinkmargin`
`tex.hoffset`	`tex.pdfpageheight`
`tex.hsize`	`tex.pdfpagewidth`
`tex.lineskiplimit`	`tex.pdfpxdimen`
`tex.mathsurround`	`tex.pdfthreadmargin`
`tex.maxdepth`	`tex.pdfvorigin`
`tex.nulldelimiterspace`	`tex.predisplaysize`
`tex.overfullrule`	`tex.scriptspace`
`tex.pagebottomoffset`	`tex.splitmaxdepth`
`tex.pageheight`	`tex.vfuzz`
`tex.pagerightoffset`	`tex.voffset`
`tex.pagewidth`	`tex.vsize`
`tex.parindent`	

Read–only:

`tex.pagedepth`	`tex.pageshrink`
`tex.pagefilllstretch`	`tex.pagestretch`
`tex.pagefillstretch`	`tex.pagetotal`
`tex.pagefilstretch`	`tex.prevdepth`
`tex.pagegoal`	

4.1.3 Direction parameters

All direction parameters are read–only and return a lua string

`tex.bodydir`	`tex.pardir`
`tex.mathdir`	`tex.textdir`
`tex.pagedir`	

4.1.4 Glue parameters

All glue parameters are read-only and return a lua string

```
tex.abovedisplayshortskip        tex.parskip
tex.abovedisplayskip             tex.rightskip
tex.baselineskip                 tex.spaceskip
tex.belowdisplayshortskip        tex.splittopskip
tex.belowdisplayskip             tex.tabskip
tex.leftskip                     tex.topskip
tex.lineskip                     tex.xspaceskip
tex.parfillskip
```

4.1.5 Muglue parameters

All muglue parameters are read-only and return a lua string

```
tex.medmuskip
tex.thickmuskip
tex.thinmuskip
```

4.1.6 Tokenlist parameters

All tokenlist parameters are read-only and return a lua string

```
tex.errhelp                      tex.everyvbox
tex.everycr                      tex.output
tex.everydisplay                 tex.pdfpageattr
tex.everyeof                     tex.pdfpageresources
tex.everyhbox                    tex.pdfpagesattr
tex.everyjob                     tex.pdfpkmode
tex.everymath
tex.everypar
```

4.1.7 Convert commands

The supported commands at this moment are:

```
tex.AlephVersion                 tex.formatname
tex.Alephrevision                tex.jobname
tex.OmegaVersion                 tex.pdfnormaldeviate
tex.Omegarevision                tex.pdftexbanner
tex.eTeXVersion                  tex.pdftexrevision
tex.eTeXrevision
```

All 'convert' commands are read-only and return a lua string

This list looks haphazard, but it really is not. These are all the cases of the 'convert' internal command that do not require an argument.

4.1.8 Count, dimension and token registers

TEX's counters (`\count`), dimensions (`\dimen`) and token (`\toks`) registers can be accessed and written to using three virtual sub-tables of the `tex` table:

```
tex.count
tex.dimen
tex.toks
```

It is possible to use the names of relevant `\countdef`, `\dimendef`, or `\toksdef` control sequences as indices to these tables:

```
tex.count.scratchcounter = 0
enormous = tex.dimen["maxdimen"]
```

In this case, luatex looks up the value for you on the fly. You have to use a valid `\countdef` (or `\dimendef`, or `\toksdef`), anything else will generate an error (the goal is to eventually also allow `<chardef tokens>` and even macros that expand into a number)

The count registers accept and return lua numbers.

The dimension registers accept lua numbers (in scaled points) or strings (with included dimension). The result is always a number in scaled points.

The token registers accept and return lua strings. Lua strings are converted to token lists using `\the\toks` style expansion.

As an alternative to array addressing, there are also accessor functions defined:

```
tex.setdimen(number n, string s)
tex.setdimen(string s, string s)
tex.setdimen(number n, number n)
tex.setdimen(string s, number n)
number n = tex.getdimen(number n)
number n = tex.getdimen(string s)

tex.setcount(number n, number n)
tex.setcount(string s, number n)
number n = tex.getcount(number n)
number n = tex.getcount(string s)

tex.settoks (number n, string s)
tex.settoks (string s, string s)
string s = tex.gettoks (number n)
string s = tex.gettoks (string s)
```

4.1.9 Box register size information

The current dimensions of `\box` registers can be read and altered using three other virtual sub-tables :

```
tex.wd
tex.ht
tex.dp
```

These are indexed strictly by number.

The box size registers accept lua numbers (in scaled points) or strings (with included dimension). The result is always a number in scaled points.

As an alternative to array addressing, there are also accessor functions defined:

```
tex.setboxwd(number n, string s)
tex.setboxwd(number n, number n)
number n = tex.getboxwd(number n)

tex.setboxht(number n, string s)
tex.setboxht(number n, number n)
number n = tex.getboxht(number n)

tex.setboxdp(number n, string s)
tex.setboxdp(number n, number n)
number n = tex.getboxdp(number n)
```

4.1.10 Print functions

The `tex` table also contains the three print functions that are the major interface from lua scripting to TEX.

The arguments to these three functions are all stored in an in-memory virtual file that is fed to the TEX scanner as the result of the expansion of `\directlua`.

The total amount of returnable text from a `\directlua` command is only limited by available system RAM. However, each separate printed string has to fit completely in TEX's input buffer.

4.1.10.1 `tex.print`

```
tex.print(<string s>, ...)
tex.print(<number n>, <string s>, ...)
```

Each string argument is treated by TEX as a separate input line.

The optional parameter can be used to print the strings using the catcode regime defined by `\catcodetable` *n*. If *n* is not a valid catcode table, then it is ignored, and the currently active cat‐code regime is used instead.

The very last string of the very last `tex.print()` command in a `\directlua` will not have the `\endlinechar` appended, all others do.

4.1.10.2 `tex.sprint`

```
tex.sprint(<string s>, ...)
tex.sprint(<number n>, <string s>, ...)
```

Each string argument is treated by TEX as a special kind of input line that makes it suitable for use as a partial line input mechanism:

- TEX does not switch to the 'new line' state, so that leading spaces are not ignored
- no `\endlinechar` is inserted
- trailing spaces are not removed

4.1.10.3 `tex.write`

```
tex.write(<string s>, ...)
```

Each string argument is treated by TEX as a special kind of input line that makes is suitable for use as a quick way to dump information:

- all catcodes on that line are either 'space' (for " ") or 'character' (for all others).
- there is no `\endlinechar` appended.

4.2 The texio library

This library takes care of the low–level I/O interface.

4.2.1 Printing functions

4.2.1.1 `texio.write`

```
texio.write(string target, tring s)
texio.write(string s)
```

Without the `target` argument, Writes the string to the same location(s) TEX writes messages to at this moment. If `\batchmode` is in effect, it writes only to the log, otherwise it writes to the log and the terminal.

The optional `target` can be one of three possibilities: 'term', 'log' or 'term and log'.

4.2.1.2 `tex.write_nl`

```
texio.write_nl(string target, tring s)
texio.write_nl(string s)
```

Like `texio.write`, but make sure that the string s will appear at the beginning of a line. You can use an empty string if you only want to move to the next line.

4.3 The pdf library

This table contains the current `h` en `v` values that define the location on the output page. The values can be queried and set using scaled points as units.

```
pdf.v
pdf.h
```

The associated function calls are

```
pdf.setv(number n)
number n = pdf.getv()
pdf.seth(number n)
number n = pdf.geth()
```

4.4 The callback library

This library has a function that is used to register callbacks, and a function that lists the registered callbacks

The callback library is only available in lua state zero (0).

```
callback.register(string <callback name>,function <callback_func>)
callback.register(string <callback name>,nil)
```

where the ⟨callback name⟩ is a predefined callback name, see below.

LUATEX internalizes the callback function in such a way that it does not matter if you redefine a function accidentally.

Callback assignments are always global. You can use the special value 'nil' instead of a function for clearing the callback.

```
table <info> = callback.list()
```

The keys in the table are the known callback names, the value is a boolean where `true` means that the callback is currently set (active).

4.4.1 File discovery callbacks

4.4.1.1 `find_read_file` and `find_write_file`

You callback function should have the following conventions:

```
string <actual_name> = function (number <id_number>, string <asked_name>)
```

Arguments:

id_number
:   zero for the log or \input files, or TeX's \read or \write number incremented by one (\read0 becomes 1).

asked_name
:   the user—supplied filename, as found by \input, or \openin, or \openout.

Return value:

actual_name
:   the filename used. For the very first file that is read in by TEX, you have to make sure you return an `actual_name` that has an extension and that is suitable for use as `jobname`. If you don't, you will have to manually fix the name for the log file and output file, and an eventual format filename will become mangled, since these depend on the jobname.

4.4.1.2 `find_output_file`

You callback function should have the following conventions:

```
string <actual_name> = function (string <asked_name>)
```

The `asked_name` is the PDF or DVI file for writing.

4.4.1.3 `find_format_file`

You callback function should have the following conventions:

```
string <actual_name> = function (string <asked_name>)
```

The `asked_name` is a format file for reading (the format file for writing is always opened in the current directory).

4.4.1.4 `find_truetype_file` and `find_type1_file`

You callback function should have the following conventions:

```
string <actual_name> = function (string <asked_name>)
```

The `asked_name` is a font file. This callback is called while LUATEX is building its internal list of needed font files, so the actual timing may surprise you. Your return value is later fed back into the matching `read__file` callback.

Strangely enough, `find_type1_file` is also used for OpenType (otf) fonts.

4.4.1.5 `find_image_file`

You callback function should have the following conventions:

```
string <actual_name> = function (string <asked_name>)
```

The `asked_name` is an image file. Your return value is used to open a file from the harddisk, so make sure you return something that is considered the name of a valid file by your operating system.

4.4.2 File reading callbacks

4.4.2.1 `open_read_file`

You callback function should have the following conventions:

```
table <env> = function (string <file_name>)
```

Argument:

file_name
    the filename returned by a previous `find_read_file` or the return value of `kpse_find_file()` if there was no such callback defined.

Return value:

env
    this is a table containing one required and one optional callback functions for this file. The required field is '`reader`' and the associated function will be called once for each new line to be read, the optional one is '`close`' that will be called once when LUATEX is done with the file. Neither function will receive an argument, so you should set you a proper closure for them.

4.4.2.1.1 `reader`

LUATEX will run this function whenever it needs a new input line from the file.

```
function ()
  return string <line>
end
```

Your function should return either a string or 'nil'. The value 'nil' signals that the end of file has occurred, and will make TEX call the optional 'close' function next.

4.4.2.1.2 close

LUATEX will optionally run this function when it needs to close the file.

```
function ()
  return
end
```

Your function should not return any value.

4.4.2.2 read_font_file

This function is called when TEX needs to read a `ofm` or `tfm` file.

```
function (string <name>)
    return boolean <success>, string <data>, number <data_size>
end
```

success
    return false when the file cannot be found.
data
    the bytes comprising the file.
data_size
    the length of the `data`, in bytes.

return an empty string and zero if the file was found but there was a reading problem.

4.4.2.3 read_vf_file

Like `read_font_file`, but for virtual fonts.

4.4.2.4 read_ocp_file

Like `read_font_file`, but for ocp files.

4.4.2.5 read_map_file

Like `read_font_file`, but for map files.

4.4.2.6 `read_enc_file`

Like `read_font_file`, but for enc files.

4.4.2.7 `read_sfd_file`

Like `read_font_file`, but for subfont definition files.

4.4.2.8 `read_miscfonts_file`

Like `read_font_file`, but for t3 font (pgc) files.

4.4.2.9 `read_pk_file`

Like `read_font_file`, but for pk bitmap files. The argument `<name>` is a bit special in this case. It's form is

```
<base res>dpi/<fontname>.<actual res>pk
```

So you may be asked for `600dpi/manfnt.720pk`. It is up to you to find a 'reasonable' bitmap file to go with that specification.

4.4.2.10 `read_data_file`

Like `read_font_file`, but for embedded files (`\pdfobj file "..."`).

4.4.2.11 `read_truetype_file`

Like `read_font_file`, but for truetype font files. The `name` is a path name as returned by `find_truetype_file` or `kpse_find_file`.

4.4.2.12 `read_type1_file`

Like `read_font_file`, but for type1 font files. The `name` is a path name as returned by `find_type1_file` or `kpse_find_file`.

4.4.2.13 `read_opentype_file`

Like `read_font_file`, but for opentype font files. The `name` is a path name as returned by `find_type1_file` or `kpse_find_file`.

4.4.3 Information reporting callbacks

4.4.3.1 `start_run`

```
function ()
```

Replaces the code that prints LuaTEX's banner

4.4.3.2 `stop_run`

```
function ()
```

Replaces the code that prints LuaTEX's statistics and 'Output written to' messages.

4.4.3.3 `start_page_number`

```
function ()
```

Replaces the code that prints the [and the page number at the begin of `\shipout`. This callback will also override the printing of box information that normally takes place when `\tracingoutput` is positive.

4.4.3.4 `stop_page_number`

```
function ()
```

Replaces the code that prints the] at the end of `\shipout`

4.4.3.5 `show_error_hook`

```
function (string <message>, string <indicator>, number <lineno>)
return end function (string <message>, number <indicator>, number
<lineno>) return end
```

This callback is run from inside the TEX error function, and the idea is to allow you to do some extra reporting on top of what TEX already does (none of the normal actions are removed).

message
  is the formal error message TEX has given to the user (the line after the "! ")
indicator
  is either a filename (when it is a string) or a location indicator (a number) that can means lots of different things like a token list id or a `\read` number.
lineno
  is the current line number

This is an investigative item only, only for 'testing the water'.

The final goal is the total replacement of TEX's error handling routines, but that needs lots of adjustments in the web source because TEX deals with errors in a somewhat haphazard fashion.

4.5 The lua library

This library contains two read-only items:

4.5.1 Variables

```
number n = lua.id
```

the id number of the instance

```
string s = lua.version
```

a luatex version identifier string (currently `"0.1"`)

4.5.2 Lua bytecode registers

Lua registers can be used to communicate lua functions across lua states. The accepted values for assignments are functions and nil. Likewise, the retrieved value is either a function or nil.

```
lua.bytecode[n] = function () .. end
lua.bytecode[n]()
```

The contents of the `lua.bytecode` array is stored inside the format file as actual lua bytecode, so it can also be used to preload lua code.

The associated function calls are

```
function f = lua.getbytecode(number n)
lua.setbytecode(number n, function f)
```

4.6 The kpse library

4.6.1 `kpse.find_file`

The most important function in the library is find_file:

```
string f = kpse.find_file(string filename)
string f = kpse.find_file(string filename, string ftype)
string f = kpse.find_file(string filename, boolean mustexist)
string f = kpse.find_file(string filename, string ftype, boolean mustexist)
```

Arguments:

filename
    the name of the file you want to find, with or without extension.
type
    maps to the '–format' argument of `kpsewhich`. The supported values are:

`"gf"`	`"TeX system documentation"`
`"pk"`	`"texpool"`
`"bitmap font"`	`"TeX system sources"`
`"tfm"`	`"PostScript header"`
`"afm"`	`"Troff fonts"`
`"base"`	`"type1 fonts"`
`"bib"`	`"vf"`
`"bst"`	`"dvips config"`
`"cnf"`	`"ist"`
`"ls-R"`	`"truetype fonts"`
`"fmt"`	`"type42 fonts"`
`"map"`	`"web2c files"`
`"mem"`	`"other text files"`
`"mf"`	`"other binary files"`
`"mfpool"`	`"misc fonts"`
`"mft"`	`"web"`
`"mp"`	`"cweb"`
`"mppool"`	`"enc files"`
`"MetaPost support"`	`"cmap files"`
`"ocp"`	`"subfont definition files"`
`"ofm"`	`"opentype fonts"`
`"opl"`	`"pdftex config"`
`"otp"`	`"lig files"`
`"ovf"`	`"texmfscripts"`
`"ovp"`	
`"graphic/figure"`	
`"tex"`	

The default type is `"tex"`.

mustexist

is similar to kpsewhich's '-must-exist', and the default is 'false'. If you specify 'true' (or a non-zero integer), then the kpse library will search the disk as well as the ls-R databases.

4.6.2 `kpse.expand_path`

Like kpsewhich's '-expand-path':

```
string r = kpse.expand_path(string s)
```

4.6.3 `kpse.expand_var`

Like kpsewhich's '-expand-var':

```
string r = kpse.expand_var(string s)
```

4.6.4 `kpse.expand_braces`

Like kpsewhich's '-expand-braces':

```
string r = kpse.expand_braces(string s)
```

4.7 The statistics library

This contains a number of run—time configuration items that you may find useful in message reporting, as well as an iterator function that gets all of the names and values as a table.

```
table <info> = statistics.list()
```

The keys in the table are the known items, the value is the current value.

Almost all of the values in `statistics` are fetched through a metatable at run—time whenever they are accessed, so you cannot use `pairs` on `statistics`, but you *can* use `pairs` on `<info>`, of course.

If you do not need the full list, you can also ask for a single item by using it's name as an index into `statistics`.

The current list is:

Key	Explanation
pdf_gone	written pdf bytes
pdf_ptr	not yet written pdf bytes
dvi_gone	written dvi bytes
dvi_ptr	not yet written dvi bytes

total_pages	number of written pages
output_file_name	name of the pdf or dvi file
log_name	name of the log file
banner	terminal display banner
pdftex_banner	–
var_used	variable (one-word) memory in use
dyn_used	token (multi-word) memory in use
str_ptr	number of strings
init_str_ptr	number of initex strings
max_strings	maximum allowed strings
pool_ptr	string pool index
init_pool_ptr	initex string pool index
pool_size	maximum allowed string characters
lo_mem_max	current top of multi-word memory
mem_min	bottom index of memory array
mem_end	top index of memory array
hi_mem_min	current bottom of one-word memory
cs_count	number of control sequences
hash_size	size of hash
hash_extra	extra allowed hash
font_ptr	number of active fonts
hyph_count	hyphenation exceptions
hyph_size	max used hyphenation exceptions
max_in_stack	max used input stack entries
max_nest_stack	max used nesting stack entries
max_param_stack	max used parameter stack entries
max_buf_stack	max used buffer position
max_save_stack	max used save stack entries
stack_size	input stack size
nest_size	nesting stack size
param_size	parameter stack size
buf_size	line buffer size
save_size	save stack size
obj_ptr	max pdf object pointer
obj_tab_size	pdf object table size
pdf_os_cntr	max pdf object stream pointer
pdf_os_objidx	pdf object stream index
pdf_dest_names_ptr	max pdf destination pointer
dest_names_size	pdf destination table size
pdf_mem_ptr	max pdf memory used
pdf_mem_size	pdf memory size
biggest_used_mark	max referenced marks class

4.8 The texconfig table

This is a table that is created empty. A startup lua script could fill this table with a number of settings that are read out by the executable after loading and executing the startup file.

key	type	default	explanation
mem_bot	number	0	cf. web2c docs
main_memory	number	250000	cf. web2c docs
extra_mem_top	number	0	cf. web2c docs
extra_mem_bot	number	0	cf. web2c docs
pool_size	number	100000	cf. web2c docs
string_vacancies	number	75000	cf. web2c docs
pool_free	number	5000	cf. web2c docs
max_strings	number	15000	cf. web2c docs
strings_free	number	100	cf. web2c docs
trie_size	number	20000	cf. web2c docs
hyph_size	number	659	cf. web2c docs
buf_size	number	3000	cf. web2c docs
nest_size	number	50	cf. web2c docs
max_in_open	number	15	cf. web2c docs
param_size	number	60	cf. web2c docs
save_size	number	4000	cf. web2c docs
stack_size	number	300	cf. web2c docs
dvi_buf_size	number	16384	cf. web2c docs
error_line	number	79	cf. web2c docs
half_error_line	number	50	cf. web2c docs
max_print_line	number	79	cf. web2c docs
ocp_list_size	number	1000	cf. web2c docs
ocp_buf_size	number	1000	cf. web2c docs
ocp_stack_size	number	1000	cf. web2c docs
hash_extra	number	0	cf. web2c docs
pk_dpi	number	72	cf. web2c docs
kpse_init	boolean	true	`false` totally disables Kpathsea initialisation (only ever unset this if you implement *all* file find callbacks!)
trace_file_names	boolean	true	`false` disables TeX's normal file open–close feedback (the assumption is that callbacks will take care of that).
src_special_auto	boolean	false	Source specials sub-item
src_special_everypar	boolean	false	Source specials sub-item
src_special_everyparend	boolean	false	Source specials sub-item
src_special_everycr	boolean	false	Source specials sub-item
src_special_everymath	boolean	false	Source specials sub-item
src_special_everyhbox	boolean	false	Source specials sub-item
src_special_everyvbox	boolean	false	Source specials sub-item
src_special_everydisplay	boolean	false	Source specials sub-item

| file_line_error | boolean | false | Do `file:line` style error messages |
| halt_on_error | boolean | false | Abort run on the first encountered error |

5 Modifications

Besides the expected changes caused by new functionality, there are a number of not–so–expected changes. These are sometimes a side–effect of a new (conflicting) feature, or, more often than not, a change necessary to clean up the internal interfaces.

5.1 Changes from T_EX 3.141592

- There is no pool file, all strings are embedded during compilation.
- "plus 1 fillll" does not generate an error. The extra 'l' is simply typeset.

5.2 Changes from ε-T_EX 2.2

- The ε-T_EX functionality is always present and enabled (but see below about T_EXX_ET), so the prepended asterisk or `-etex` switch for initex is not needed.
- T_EXX_ET is not present, so the primitives

  ```
  \TeXXeTstate
  \beginR
  \beginL
  \endR
  \endL
  ```

  are missing

5.3 Changes from PDFT_EX 1.40

- A number of 'utility functions' is removed:

  ```
  \pdfelapsedtime
  \pdfescapehex
  \pdfescapename
  \pdfescapestring
  \pdffiledump
  \pdffilemoddate
  \pdffilesize
  \pdflastmatch
  \pdfmatch
  \pdfmdfivesum
  \pdfresettimer
  \pdfshellescape
  ```

```
\pdfstrcmp
\pdfunescapehex
```

- A few other experimental primitives are provided without the extra 'pdf' prefix, so they are simply called:

```
\primitive
\ifprimitive
\ifabsnum
\ifabsdim
```

5.4 Changes from ALEPH RC4

- The input translations from ALEPH are not implemented, the related primitives are not available

```
\DefaultInputMode
\noDefaultInputMode
\noInputMode
\InputMode
\DefaultOutputMode
\noDefaultOutputMode
\noOutputMode
\OutputMode
\DefaultInputTranslation
\noDefaultInputTranslation
\noInputTranslation
\InputTranslation
\DefaultOutputTranslation
\noDefaultOutputTranslation
\noOutputTranslation
\OutputTranslation
```

- A small series of bounds checking fixes to `\ocp` and `\ocplist` has been added to prevent the system from crashing due to array indexes running out of bounds.
- The `\hoffset` bug when `\pagedir TRT` is fixed, removing the need for an explicit fix to `\hoffset`
- A bug causing `\fam` to fail for family numbers above 15 is fixed.
- Some bits of ALEPH assumed `0` and `null` were identical. This resulted for instance in a bug that sometimes caused an eternal loop when trying to `\show` a box.
- A fair amount of minor bugs are fixed as well, most of these related to `\tracingcommands` output.
- The number of fonts, ocps and ocplists is capped off at about 8% of their maximum ALEPH value (around 5000 for each).

5.5 Changes from standard WEB2C

- There is no mltex
- There is no enctex
- The following command–line switches are silently ignored:

  ```
  -8bit
  -translate-file=TCXNAME
  -mltex
  -enc
  -etex
  ```

- \openout whatsits are not written to the log file.
- Some of the so—called web2c extensions are hard to set up in non–kpse mode because texmf.cnf is not read: shell-escape is off (but that is not a problem because of Lua's os.execute), and the paranoia checks on openin and openout do not happen (however, it is easy for a Lua script to do this itself by overloading io.open).

6 Implementation notes

1 Primitives overlap

The primitives

```
\pdfpagewidth and \pagewidth,
\pdfpageheight and \pageheight,
\fontcharwd and \charwd,
\fontcharht and \charht,
\fontchardp and \chardp,
\fontcharic and \charic,
```

are all aliases of each other.

2 Sparse arrays

The `\mathcode`, `\delcode`, `\catcode`, `\sfcode`, `\lccode` and `\uccode` tables are now sparse arrays that are implemented in C. They are no longer part of the TEX "equivalence table" and because each had 1.1 million entries with a few memory words each, this makes a major difference in memory usage.

These assignments do not yet show up when using the etex tracing routines `\tracingassigns` and `\tracingrestores` (code simply not written yet)

A side-effect of the current implementation is that `\global` is now more expensive in terms of processing than non-global assignments.

See `mathcodes.c` and `textcodes.c` if you are interested in the gory details.

3 Simple single-character csnames

Single-character commands are no longer treated special in the internals, they are stored in the hash just like the multiletter csnames.

The code that displays control sequences explicitly checks if the length is one when it has to decide whether or not to add a trailing space.

4 Compressed format

The format is passed through zlib, allowing it to shrink to roughly a third of the size it would have had in uncompressed form. This takes a bit more CPU cycles but much less disk I/O, so it should still be faster.

The chosen compression factor is fairly low, equivalent to `gzip -3`.

5 Binary file reading

All of the internal code is changed in such a way that if one of the `read_xxx_file` callbacks is not set, then the file is read by a C function using basically the same convention as the callback: a single read into a buffer big enough to hold the entire file contents. While this uses more memory than the previous code (that mostly used `getc` calls), it can be quite a bit faster (depending on your I/O subsystem).

7 Known bugs

- Under some conditions, the **autoexpand** font feature can crash the executable.
- **Hyphenation** can only deal with the Base Multilingual Plane (BMP)
- There are (sometimes?) UTF-8 encoded UTF-8 bytes in the pseudo-buffer lines that are shown during error()

8 TODO

On top of the 'normal' extensions that are planned, there are some more specific small feature requests
.

- Low-level support is still missing but needed to escape from the UTF-8 regime when dealing with
  `\special` and `\pdfliteral`.
- Implement the TEX primitive `\dimension`, cf. `\number`
- Change the lua table typetex.dimen to accept and return float values instead of strings
- Do something about `\withoutpt` and/or a new register type `\real`?
- Implement the TEX primitive `\htdp`?
- Do boxes with dual baselines.
- A way to (re?)calculate the width of a `\vbox`, taking only the natural width of the included items
  into account.