Lua & TEX tokens

Taco Hoekwater

May 3, Bachotek

X14

LuaTEX has had a token Lua library since the early beginnings, but it was more a proof of concept, and it has never worked really well at that.

This talk presents a new, hopefully better interface between Lua code and the TEX language parsing.

X14

Old state

To get a new token from the input, you called the function `token.get_next()` or `token.lookup()`:

```
local l = token.lookup("if")
```

Such ‘tokens’ were simple Lua tables with three integer values within:

- `l[1]` The command code → 120
- `l[2]` The command modifier code → 0
- `l[3]` The control sequence id → 65536

Use

To get something meaningful out of those numerical values, you had to run another function. For example:

```
token.command_name(l) → "if_test"
token.csname_name(l) → "iftrue"
token.is_expandable(l) → true
token.is_activechar(l) → false
token.is_protected(l) → false
```

Other functions

```
token.create(<number> chr [,<number cmd])
token.csname_id(<string> csname)
token.command_id(<string> cmdname)
token.expand()
```

Planned new state

The functions `get_next()`, `lookup()`, and `create()` still exist, but they return a userdata object that contains the *actual* TEX token. Some of the helper functions go away, and instead that are accessible fields in the token itself:

```
l.cmd
l.mod
l.cs
l.cmdname
l.csname
l.expandable
l.active
l.protected
```

New functions

Various new functions are for actual input parsing:

```
token.scan_keyword(<string> keyword)
token.scan_int()
token.scan_dimen()
token.scan_glue()
token.scan_toks()
```

To think about

- `token.expand()` behaviour
- `\meaning` and `\def`
- `l.next` or actual tables
- more `scan_xxxx()` functions
- input stack

X14