Lua & TEX tokens

Taco Hoekwater

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, better interface between Lua code and the TEX language parsing.

1. Introduction

When LuaTEX development started, the very first command added was `\directlua`, providing a way to run Lua code in the middle of a TEX document. This allows some pretty powerful processing to be done, but on its own, that is not enough to actually influence the handling of the TEX document input.

Just as LuaTEX has built-in lua extensions to handle things like node processing a font loading, it also has a built-in library intended to manipulate TEX tokens. Unfortunately, this `token` library was one of the first ones written, and it clearly suffers from the lack of understanding of the internals of both TEX and Lua that we had at that time.

2. Issues with old library

The main problem with the existing library is that it does not treat TEX tokens as proper Lua objects. The interface is based on a small Lua array that contains three numerical values, for example:

```
\directlua{
  local t = token.get_next()
  table.print(t)
}!
```

This prints

```
t={
 12,
 33,
 0,
}
```

which means that the next token has command code `12` (other character), the modifier was `33` (that is the numerical representation for the `!` character), and the symbol table entry was `0` (which roughly means that it was not a control sequence)

When we change the `!` into `\relax`, the output becomes:

```
t={
 0,
 1114112,
 3380,
}
```

where the internal command code is `0` (relax), the internal modifier is `1114112`, and the original string representation of the command's name can be found via symbol entry number `3380`.

The library also provides some helper functions to convert these numerical values into strings, so at least the user does not have to remember all those actual numbers. Even so, it is not a nice interface at all.

3. Plans for the new library

The new approach will be to change the internal representation of a token such that the Lua value is in fact an object pointing to the actual TEX token. This will allow a better interface for the user of the library. The result of `get_next()` will be a so-called 'userdata object', and its content will be accessed using named fields, like this:

```
\directlua{
  local t = token.get_next()
  print(t.cmdname)
  print(t.mod)
}!
```

That is the plan for the token object interface itself. There will also be functions to create and copy tokens, as well as an interface to the TEX scanner so that instead of `get_next()` it will be possible to use e.g. `scan_dimen()` to scan a dimension from the input file, and `scan_keyword()` to check for a fixed string.

Currently, this new library is in the development stage. We are adding in some functionality that we are certain will be needed, but we are also open to feature requests and discussions.