The luaimmediate package*

Marcel Krüger
tex@2krueger.de

April 11, 2020

Overwrite `\immediate` with a Lua wrapper which allows commands defined by `\luadef` to react to the presence of `\immediate`.

If you have a command `\example` defined by `\luadef\example42`, then you can allow it to appear after `\immediate` by writing

```
\directlua {require'luaimmediate'('example', true)}
```

Then the first argument to the corresponding Lua function will be `"immediate"` if the command is used after `\immediate`. By passing a function instead of `true`, an independent replacement function can be used instead specifically for use in an `\immediate` context.

1 The implementation

1.1 Lua module

Now we can define our main Lua module:

```
local extension_cmd = token.command_id'extension'
local lua_function_call_cmd = token.command_id'lua_function_call'
local lua_call_cmd = token.command_id'lua_call'
local immediate_code = 4
local minus_token = token.new(0x2D, 12)
local immediate_token = token.new(immediate_code, extension_cmd)

local scan_token = token.scan_token
local put_next = token.put_next

local immediate_lookup = {}

local function new_immediate()
  local token = scan_token()
  local tok = (token.command << 21) | token.mode
  immediate_function = immediate_lookup[tok]
```

*This document corresponds to luaimmediate v0.0.1, dated 2020/04/09.

```
  if immediate_function then
    return immediate_function()
  else
    put_next(immediate_token, token)
  end
end
local functions = lua.get_functions_table()
local immediate_func = luatexbase
                       and luatexbase.new_luafunction'immediate'
                       or (string.unpack(">I", 'imm') & 0x1FFFFF)
functions[immediate_func] = new_immediate
token.set_lua('immediate', immediate_func, 'protected', 'global')
```

The next technically is not necessary, but it does not hurt either.

```
immediate_lookup[token.new(immediate_func, lua_call_cmd).tok]
    = new_immediate

return function(id, func)
  if not token.is_token(id) then
    local tid = type(id)
    if tid == 'string' then
      if not token.is_defined(id) then
        error('Attempt to define an immediate meaning to an \z
          undefined csname.')
      end
      id = token.create(id)
    elseif tid == 'number' then
      id = math.tointeger(id)
      if (not id) or id <= 0 then
        error('Invalid number passed to luaimmediate.')
      end
      id = token.new(id, lua_call_cmd)
    else
      error('Invalid argument passed to luaimmediate.')
    end
  end
  local idtok = (id.command << 21) | id.mode
  if func == nil then
    return immediate_lookup[idtok]
  elseif func == true then
    if id.command ~= lua_call_cmd then
      error[[Explicit function required for weird immediate tokens.]]
    end
```

Now we cache `i`, not `functions[i]`. Therefore we pick up changes to `functions[i]` at some later point. That is by design, if you do want to avoid it, just pass a function instead.

```
    local i = id.mode
    func = function()
      return functions[i]'immediate'
```

```
    end
  end
  immediate_lookup[idtok] = func or nil
end
```

1.2 TEX support package

Most of the time this TEX helper is not needed, but sometimes it can be useful to ensure `\immediate` is redefined early enough.

```
⟨*package⟩
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage
  {luaimmediate}
  [2020/04/09 v0.0.1 Allow Lua commands to react to immediate]
⟨/package⟩
```

Only LuaLATEX is supported. For other engines we show an error.

```
\ifx\directlua\undefined
```

```
⟨*tex-package⟩
  \begingroup
    \ifx\PackageError\undefined
      \def\PackageError#1#2#3{\errhelp{#3}\errmessage{#1: #2}}
    \fi
⟨/tex-package⟩
```

```
    \PackageError{luaimmediate}{LuaLaTeX required}%
    {luaimmediate requires LuaLaTeX.
     Maybe you forgot to switch the engine in your editor?}
```

```
⟨*tex-package⟩
  \endgroup
⟨/tex-package⟩
```

```
\fi
```

The actual functionality is not that interesting: We just load the Lua module.

```
\directlua{require'luaimmediate'}
\endinput
```