ECMAscript

Just because it can be done

Hans Hagen

Why oh why?

We use a `mupdf` based PDF viewer: `SumatraPDF`, and occasionally we use the tools that come with `mupdf`. So when checking if that viewer supports JavaScript[1] in widgets, I noticed the standalone interpreter, which made me wonder how easy it would be to interface with it.

ECMAscript support for LuaMetaTEX uses the lightweight library subsystem; like ffi, the library interface is setup dynamically. Support is *not* integrated in LuaMetaTEX, so there are no overheads or dependencies.

We assume that the library is on the system, and when not, then there is also no support. We stick to the absolute minimum of interfacing needed and delegate everything else to Lua. We assume a stable API, and if not, well ... sorry.

The components

The optional delayed loading interface adds only a few kilobyte to LuaMetaTEX. The Lua library interface is part of the ConTEXt distribution, which means that it's officially supported. There is a TEX module that loads the lot and provides the user interface.

And of course, somewhere on the system, there should be the `mujs` library.[2]

A module like this should conform to the ConTEXt LMTX standards (a minimalistic, non-bloated API, interfacing in Lua and TEX, etc.).

In ConTEXt libraries go into the platform tree, like:

```
/tex/texmf-win64/bin/lib/luametatex/mujs/libmujs.dll
/tex/texmf-linux-64/bin/lib/luametatex/mujs/libmujs.so
/tex/texmf-osx-64/bin/lib/luametatex/mujs/libmujs.so
```

[1] The official name is ECMAscript which is the standardized core language.
[2] Taco compiled the library for his system during the talk and confirmed that it also works out of the box on MacOS. Hint: `make shared`

An example

```
\usemodule[ecmascript]

\ecmacode {
    console("");
    console("When you see this, the loading has succeeded!");
    console("");
}

\ecmacode {texprint("Just a {\\bf short} sentence.")}

\startecmacode
    texprint("And this is \\inframed{\\bs a bit longer}
sentence.")
\stopecmacode
```

Just a **short** sentence.

And this is ***a bit longer*** sentence.

Catcodes

As with the Lua interface, catcode regimes are supported:

```
\ecmacode {texprint(catcodes.vrb,"Just a {\\bf short} sentence.")}
```

Just a {\bf short} sentence.

Possible values are:

`tex` regular TEX catcode regime
`ctx` standard ConTEXt catcode regime
`vrb` verbatim catcode regime
`prt` protected ConTEXt catcode regime

Print whatever you want

```
\startecmacode
  console("We're doing some MetaPost!");
  texsprint(
      "\\startMPcode "
    + 'fill fullsquare xyscaled (6cm,1cm) withcolor "darkgray";'
    + 'fill fullsquare xyscaled (4cm,1cm) withcolor "middlegray";'
    + 'fill fullsquare xyscaled (2cm,1cm) withcolor "lightgray";'
    + "\\stopMPcode "
  );
\stopecmacode
```

Of course the code doesn't look pretty, but it can serve as a step-up to the real deal: coding in ConTEXt speak (or Lua).

Files

Because the interpreter is pretty bare, interfacing to the file system has to be provided, but we can just use what we already have (controlled by Lua).

```
\startecmacode
  var f = File("\jobname","r");
  var l = f.read("*a");
  f.close();
  texprint(
      "This file has "
    + l.length // or: l.length.toString()
    + " bytes!"
  )
\stopecmacode
```

This file has 2545 bytes!

We support the usual arguments, like `*a`, `*l`, a number indicating the bytes to read etc. There is no support for writing files (let's use the security excuse).

A file with some script:

```
function filesize(name) {
    var f = File(name,"r");
    if (f != undefined) {
        var l = f.seek("end");
        f.close();
        return l;
    } else {
        return 0;
    }
}
```

Loading that file:

```
\ecmafile{context-2020-ecmascript.js}
```

Using that function:

```
\ecmacode{texsprint("This file has " + filesize("\jobname.tex") +
" bytes!")}
```

This file has 2548 bytes!

ECMAscript from Lua

```
\startluacode
  optional.loaded.mujs.execute [[
    var MyMax = 10; // an example of persistence
  ]]

  optional.loaded.mujs.execute [[
    texsprint("\\startpacked");
    for (var i = 1; i <= MyMax; i++) {
      texprint(
        "Here is some rather dumb math test: "
      + Math.sqrt(i/MyMax)
      + "!\\par"
      );
    }
    texsprint("\\stoppacked");
  ]]
\stopluacode
```

The result:

Here is some rather dumb math test: 0.31622776601683796!
Here is some rather dumb math test: 0.4472135954999579!
Here is some rather dumb math test: 0.5477225575051661!
Here is some rather dumb math test: 0.6324555320336759!
Here is some rather dumb math test: 0.7071067811865476!
Here is some rather dumb math test: 0.7745966692414834!
Here is some rather dumb math test: 0.8366600265340756!
Here is some rather dumb math test: 0.8944271909999159!
Here is some rather dumb math test: 0.9486832980505138!
Here is some rather dumb math test: 1!

So what good is it?

Not that much value is added compared to what we already have but at least we can say that we can ECMAscript (a.k.a. JavaScript). It might convince (new) users to use the Lua interfaces, so we pay a low price and no overhead anyway.