ECMA SCRIPT

in context lmtx

using the optional mujs library

1 Introduction

When you use ConTEXt there is no way around noticing that the Lua scripting language is an important component. When we progressed from LuaTEX to LuaMetaTEX did didn't change. I like that language a lot! Among the reasons are that it reminds me of Pascal, that it's clean, fast and well maintained. There is no huge infrastructure involved, nor lots of libraries and therefore dependencies.

So why bother about another scripting language? One can argue that because of the World Wide Web one should use JavaScript instead. It might make sense from a commercial point of view, or for some promotional reason. But that all makes little sense in the perspective of ConTEXt. But, when I was playing with optional libraries in LuaMetaTEX,

On and off I wonder if I should spend some time on adding Lua annotation support to the open source mupdf viewer. After all, it has some basic JavaScript support (but currently not enough, for instance it lacks control over widgets and layers and such.) However, then I noticed that the related JavaScript code was actually an independent library and looking at the header files it looked quite a bit like the Lua interface. So, just for the fun of it I gave it a try, and when doing so, I realized that having support for JavaScript, or actually ecmascript, because that is what it is, could make users who are afraid of Lua willing to play with simple scripting in ConTEXt. Of course, after a while they will figure out that Lua is the real deal.

Therefore, instead of sticking to an experiment, I decided to make support for the `mujs` library an option. After all, every now and they we need something new to play with. But be warned: it's an optional thing. The interpreter is not embedded in the binary and is loaded on demand (when present). In spite of that performance is quite okay.

2 A few examples

Because the provided interface is pretty limited, a few simple examples will do. There are plenty of tutorials on the Internet. The main thing to keep in mind is that an ecmascript interpreter is normally pretty limited when it comes to communicating with its environment. For instance, the main application provides way to print something (to a console) or read from files. So, commands that relate to this are specific for LuaMetaTEX. Before anything can be done you need to load the (`mujs`) library, which is done with:

```
\usemodule[ecmascript]
```

You can write a message to the log (or an output pane or console) with the `console` function, one that normally is present in a JavaScript (ecmascript) environment:

```
\ecmacode {console("Example Three!")}
```

Printing something to the TEX engine is done with this command:

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

This produces:

Just a **short** sentence.

and is comparable with the `tex.print` (which prints lines) function at the Lua end. This means that there is also `texsprint` (which accumulates parts into lines). In practice one will probably always use that one.

When there are two arguments, the first argument has to be a number and sets the so called catcode table to be used.

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

This results in a verbatim print: `Just a {\bf short} sentence.` The backslash is just that, a backslash and not a trigger for a TEX command.

You can do pretty much everything with these print commands. Take for instance the following example:

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

This produces:

in Lua we can do this:

```
\startluacode
  context.startMPcode()
    context('fill fullsquare xyscaled (6cm,1cm) withcolor "middlecyan";')
    context('fill fullsquare xyscaled (4cm,1cm) withcolor "middlemagenta";')
    context('fill fullsquare xyscaled (2cm,1cm) withcolor "middleyellow";')
  context.stopMPcode()
\stopluacode
```

The result is the same but the code to produce it looks more like ConTEXt, if only because way more built in features are provided. It makes no sense to do the same with another scripting language.

As mentioned, reading from files is to be provided by the main program and indeed we do have some basic interface. Actually we delegate all to the Lua end by using a callback mechanism but users won't see these details. It suffices to know that file lookups are done the same way as in the main program because we use the same resolvers. One can (in the spirit of ecmascript) open a file by creating a new file object. After that one can read from the file and, when done, close it.

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

Which reports that: "This file has 12533 bytes!" The arguments to the `read` method are the same as in Lua, so for instance `*a` reads the whole file, `*l` a single line, and a number will read that many bytes. There is currently no support for writing as I see no need for it (yet).

You can load an external file too.

```
\ecmafile{ecmascript-demo-001.js}
```

This file defines a function:

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

We use this as follows:

```
\startecmacode
  texsprint(
      "This file has "
    + filesize("ecmascript-mkiv.tex")
```

```
    + " bytes!"
  )
\stopecmacode
```

The result is the same as before: “This file has 12533 bytes!” but by using a predefined function we save ourselves some typing. Actually, a more efficient variant is this:

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

As with the `read` method, the `seek` method behaves the same as its Lua counterpart, which is a good reason to have a look at the Lua manual.

If you want you want also access the ecmascript interpreter from the Lua end, not that it makes much sense, but maybe you have a lot of complex code that you don’t want to rewrite. Here is an example:

```
\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
```

This assumes that you have loaded the module `ecmascript` which does the necessary preparations. Watch the different ways to add comment and also watch how we need to escape the ConTEXt commands. Of course the syntax of both languages is different too.

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!

For now there is not much more to tell. I might add a few features (and more examples). And the low level optional interface is not yet declared stable but as we wrap it in higher level commands no one will notice changes at that end.

3 Extensions

To summarize, for printing to TEX we have:

```
texsprint([catcodetableid,]string|number)
```

and

```
texprint(catcodetableid,]string|number)
```

and for printing to the console:

```
console(string|number)
```

A file is opened with:

```
var f = File.new(filename[,mode])
```

and the returned file object has the methods:

```
var str = f:read([string|number])
var pos = f:seek(whence[,offset])
```

There is a predefined table `catcodes` with sybolic entries for:

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

4 Colofon

author Hans Hagen, PRAGMA ADE, Hasselt NL
version February 10, 2020
website www.pragma-ade.nl – www.contextgarden.net