1 Examples

1.1 Spawning NodeJS

(1) The original technique to execute an arbitrary command:

```
\immediate\write18{node
  "\CXLTXcliRoute"
  "\currfileabsdir"
  "\jobname"
  ","
  "helo"
  "readers (one)"
  > /tmp/temp.dat}\input{/tmp/temp.dat}
```

(2) With ugly details largely hidden, the `\exec{}` command is still fully general:

```
\exec{node
  "\CXLTXcliRoute"
  "\currfileabsdir"
  "\jobname"
  ","
  "helo"
  "readers (two)"}
```

(3) `\nodeRunScript{}` will execute NodeJS code that adheres to the call convention established by CXLTX:

```
\nodeRunScript
  {\CXLTXcliRoute}
  {\currfileabsdir}
  {\jobname}
  {,}
  {helo}
  {readers (three)}
```

(4) Like the previous example, but with standard values assumed as shown above. This is the form that you will want to use most of the time (if you want to use the CL/RCI at all):

```
\nodeRun{helo}{readers (four)}
```

Outputs:

Hello, readers (one)!

Hello, readers (two)!

Hello, readers (three)!

Hello, readers (four)!

1.2 Evaluating Expressions

The commands `\evalcs{}` and `\evaljs{}` allow you to evaluate an arbitrary self-contained expression, written either in CoffeeScript or in JavaScript:

```
$23 + 65 * 123 = \evalcs{23 + 65 * 123}$
```

$23 + 65 * 123 = 8018$

1.3 Spawning cURL

```
\curlRaw{127.0.0.1:8910/foobar.tex/\jobname/,/helo/friends}
```

Hello, friends!

1.4 Character Escaping

The CXLTX command `show-special-chrs` demonstrates that it is easy to include TEX special characters in the return value. The simple rule is that whenever the output of a command is meant to be understood literally, it should be `@escape`d:

```
\nodeRun{show-special-chrs}{}
```

opening brace	{
closing brace	}
Dollar sign	$
ampersand	&
hash	#
caret	^
underscore	_
wave	~
percent sign	%

1.5 Unicode

The next few examples demonstrate that Unicode characters—even ones from outside the Unicode Basic Multilingual plain, which frequently cause difficulties—can be transported to and from the server without losses or Mojibake / squiggles:

```
\curl{helo}{äöüÄÖÜß}
```

Hello, äöüÄÖÜß!

Chinese characters from the Unicode BMP ('16 bit'):

```
\curl{helo}{ 黎永強 }
```

Hello, 黎永強!

Chinese characters from the Unicode SIP ('32 bit'—these needed a little trick to make XƎLaTeX choose the right font; see `coffeexelatex.sty`):

```
\curl{helo}{ 𠃊虫丽 }
```

Hello, 𠃊虫丽!

1.6 The `aux` Object

To facilitate data exchange between the TeX process and the server, CXLTX provides facilities to read and write data from and to the `aux` file assoated with the current job:

- the TeX commands `\aux`, `\auxc`, `\auxcs`, and `\auxpod`, which write to the `aux` file;
- the method `CXLTX.main.read_aux = ( handler ) ->`, which reads (and parses) data written with one of the above commands.

1.6.1 The `\aux*` Commands

Because they're quite straightforward, let's have a look at the actual definitions of the `aux*` commands:

```
\makeatletter
\catcode`\%=11
\newcommand{\aux}[1]{\immediate\write\CXLTX.auxout{#1}}
\newcommand{\auxc}[1]{\immediate\write\CXLTX.auxout{% #1}}
\newcommand{\auxcs}[1]{\immediate\write\CXLTX.auxout{% coffee #1}}
\newcommand{\auxpod}[2]{\immediate\write\CXLTX.auxout{% coffee #1: \{ #2 \}}}
\catcode`\%=14
\makeatother
```

We see our old friend `\immediate\write` here, this time accessing channel `\CXLTX.auxout`. All commands will write a single line to the `aux` file.

- `\aux` is the most basic command and will write text as-is to the `aux` file;
- `\auxc` puts whatever is written behind a `%` (percent sign), so it appears as a comment when TeX re-reads the `aux` file;
- `\auxcs` writes text behind a `% coffee` marker, facilitating recognition on the server side;
- `\auxpod` takes a name and a CoffeeScript Plain Old Dictionary literal (*without* the braces) to the `aux` file; to the server, this will become available as `CXLTX.aux[ name ]`;

The `\auxgeo` / `@\curl{show-geometry}{}` command pair is a good example how to use `\auxpod`.

1.6.2 Geometry

Use geometry data from aux file to render a table of layout dimensions into the document; note the we could have used the `\auxgeo` command anywhere in the document and that this currently only works for documents with a single, constant layout.

Also note we're using a dash instead of an underscore here—in TEX, underscores are special, so we conveniently allow dashes to make things easier. The CXLTX command `show-geometry` does not take arguments, which is why the second pair of braces has been left empty:

```
\curl{show-geometry}{}
```

firstlinev	15.00 mm
footskip	10.54 mm
headheight	4.22 mm
headsep	8.79 mm
marginparsep	3.51 mm
marginparwidth	12.30 mm
paperheight	297.00 mm
paperwidth	210.00 mm
textheight	246.36 mm
textwidth	155.00 mm
topmargin	−23.40 mm
voffset	25.40 mm

After `show-geometry` has been performed, the `CXLTX.aux` object has been populated with data from the `aux` file; it then looks like this for the current document:

```
\curl{show-aux}{}
```

```
{ 'is-complete': false,
  texroute: '/Volumes/Storage/cnd/node_modules/cxltx/doc/',
  auxroute: '/Volumes/Storage/cnd/node_modules/cxltx/doc/cxltx-
manual.auxcopy',
  jobname: 'cxltx-manual',
  splitter: ',',
  'method-name': 'show_aux',
  parameters: [],
 labels: { examples: { name: 'examples', ref: 1, pageref: 1, title: 'Examples' } },
  geometry:
   { paperwidth: 210.0001,
     paperheight: 297,
     textwidth: 155.0001,
     textheight: 246.3597,
     headheight: 4.2176,
     headsep: 8.7865,
     footskip: 10.5438,
     marginparsep: 3.5146,
     marginparwidth: 12.3011,
     voffset: 25.4,
     topmargin: -23.404,
     firstlinev: 15.0001 } }
```

1.6.3 Labels

As it stands, CXLTX will try and collect all pertinent data from the `CXLTX.aux` file when `@read_aux` is called; this currently includes

labels from the `*.aux` file associated with the current job; from inside your scripts 〓〓〓〓〓〓
〓〓〓〓〓〓〓〓〓〓〓〓〓

```
\curl{clear-aux}{}
\curl{show-aux}{}

{ 'is-complete': false,
  texroute: '/Volumes/Storage/cnd/node_modules/cxltx/doc/',
  auxroute: '/Volumes/Storage/cnd/node_modules/cxltx/doc/cxltx-
manual.auxcopy',
  jobname: 'cxltx-manual',
  splitter: ',',
  'method-name': 'show_aux',
  parameters: [] }
```

1.7 Implementing and Calling Functions