1 Introduction

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

```
\immediate\write18{node
  "\CXmainroute"
  "\currfilepath"
  "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
  "\CXmainroute"
  "\currfilepath"
  "helo"
  "readers (two)"}
```

(3) `\noderunscript{}` will execute NodeJS code that adheres to the call convention established by *Coffee*XeLaTeX:

```
\noderunscript
  {\CXmainroute}
  {\currfilepath}
  {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:

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

Outputs:

Hello, readers (one)!

Hello, readers (two)!

Hello, readers (three)!

Hello, readers (four)!

2 Configuration

To use *Coffee*XeLaTeX, put

```
\usepackage{coffeexelatex}
```

into the header section of your LaTeX file.

You may also want to include these lines in your LaTeX document; these define, respectively, the route to the *Coffee*XeLaTeX executable (`coffeexelatex/lib/main.js`) and the temporary file that is used to communicate between TeX and your scripts (relative routes are resolved with respect to the current working directory, so if you set a relative route, you must always run TeX from within the same directory):

```
\renewcommand{\CXmainroute}{../../lib/main}
\renewcommand{\CXtempoutroute}{/tmp/coffeexelatex.tex}
```

3 Character Escaping

The *Coffee*XeLaTeX 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:

Command:

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

Output:

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

4 The aux Object

4.1 Labels

*Coffee*XeLaTeX will try and collect all labels from the `*.aux` file associated with the current job; from inside your scripts ▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬

Command:

```
\noderun{show-aux}{}
```

Output:

```
{ 'is-complete': false,
  texroute: 'example-1.tex',
  'method-name': 'show_aux',
  parameters: [],
  auxroute: 'example-1.auxcopy',
  labels:
   { intro: { name: 'intro', ref: 1, pageref: 2, title: 'Introduction' },
     config: { name: 'config', ref: 2, pageref: 3, title: 'Configuration' },
     esc: { name: 'esc', ref: 3, pageref: 4, title: 'Character Escaping' },
     curl: { name: 'curl', ref: 5, pageref: 6, title: 'Curl' } } }
```

4.2 Evaluating Expressions

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

Command:

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

Output:

$23 + 65 * 123 = 8018$

5 Curl

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

Hello, friends!

5.1 The URL environment

Typing URLs in LaTeX can be quite a chore, given the number of active and otherwise 'special' characters to take care of: not only does TeX itself define some special characters, not only do the RFCs that govern URL syntax consider ▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬ special—when we communicate with our *Coffee*XeLaTeX server, we do so by executing a 'curl ...' command via the OS shell (normally `sh`), which again has its own rich set of specials. In order to alleviate the burden on the casual user, we define a new environment, 'URL', that somewhat simplifies writing (parts of) URLs:

```
\begin{URL}
\curl{helo}{`\{ [ $ ~ % \# ^ | \} ] '?}
\end{URL}
```

% Hello, URI malformed!