introduction

August 18, 2006

Contents

1 Introduction

ParenScript is a simple language that looks a lot like Lisp, but actually is JavaScript in disguise. Actually, it is JavaScript embedded in a host Lisp. This way, JavaScript programs can be seamlessly integrated in a Lisp web application. The programmer doesn't have to resort to a different syntax, and JavaScript code can easily be generated without having to resort to complicated string generation or `FORMAT` expressions.

An example is worth more than a thousand words. The following Lisp expression is a call to the ParenScript "compiler". The ParenScript "compiler" transforms the expression in ParenScript into an equivalent, human-readable expression in JavaScript.

```
(js
  (defun foobar (a b)
    (return (+ a b))))
```

The resulting javascript is:

```
"
function foobar(a, b) {
  return a + b;
}
"
```

Great care has been given to the indentation and overall readability of the generated JavaScript code.

2 Features

ParenScript supports all the statements and expressions defined by the EcmaScript 262 standard. Lisp symbols are converted to camelcase, javascript-compliant syntax. This idea is taken from Linj by Antonio Menezes Leitao. Here are a few examples of Lisp symbol to JavaScript name conversion:

```
(js-to-string 'foobar)      => "foobar"
(js-to-string 'foo-bar)     => "fooBar"
(js-to-string 'foo-b-@-r)   => "fooBAtR"
(js-to-string 'foo-b@r)     => "fooBatr"
(js-to-string '*array)      => "Array"
(js-to-string '*math.floor) => "Math.floor"
```

It also supports additional iteration constructs, relieving the programmer of the burden of iterating over arrays. `for` loops can be written using the customary `DO` syntax.

```
(js
  (do ((i 0 (incf i))
       (j (aref arr i) (aref arr i)))
      ((>= i 10))
    (alert (+ "i is " i " and j is " j))))

; compiles to
"
for (var i = 0, j = arr[i]; i < 10; i = ++i, j = arr[i]) {
  alert('i is ' + i + ' and j is ' + j);
}
"
```

ParenScript uses the Lisp reader, allowing for reader macros. It also comes with its own macro environment, allowing host macros and ParenScript to coexist without interfering with each other. Furthermore, ParenScript uses its own compiler macro system, allowing for an even further customization of the generation of JavaScript. For example, the `1+` construct is implemented using a ParenScript macro:

```
(defjsmacro 1+ (form)
  `(+ ,form 1))
```

ParenScript allows the creation of JavaScript objects in a Lispy way, using keyword arguments.

```
(js
  (create :foo "foo"
          :bla "bla"))

; compiles to
"
{ foo : 'foo',
  bla : 'bla' }
"
```

ParenScript features a HTML generator. Using the same syntax as the `HTMLGEN` package of Franz, Inc., it can generate JavaScript string expressions. This allows for a clean integration of HTML in ParenScript code, instead of writing the tedious and error-prone string generation code generally found in JavaScript.

```
(js
  (defun add-div (name href link-text)
    (document.write
     (html ((:div :id name)
            "The link is: "
            ((:a :href href) link-text))))))

; compiles to
"
function addDiv(name, href, linkText) {
  document.write('<div id=\"' + name + '\">The link is: <a href=\"'
                                 + href + '\">'
                                 + linkText + '</a></div>');
}
"
```

In order to have a complete web application framework available in Lisp, ParenScript also provides a sexp-based syntax for CSS files. Thus, a complete web application featuring HTML, CSS and JavaScript documents can be generated using Lisp syntax, allowing the programmer to use Lisp macros to factor out the redundancies and complexities of Web syntax. For example, to generate a CSS inline node in a HTML document:

```
(html-stream *standard-output*
  (html
   (:html
    (:head
     (css (* :border "1px solid black")
          (div.bl0rg :font-family "serif")
          (("a:active" "a:hoover") :color "black" :size "200%"))))))

; which produces

<html><head><style type="text/css">
<!--
* {
   border:1px solid black;
}

div.bl0rg {
   font-family:serif;
}

a:active,a:hoover {
   color:black;
   size:200%;
}

-->
```

```
</style>
</head>
</html>
```

3 Getting ParenScript

ParenScript can be obtained from the BKNR subversion repository at

```
svn://bknr.net/trunk/bknr/src/js
```

ParenScript does not depend on any part of BKNR though. You can download snapshots of ParenScript at the webpage

```
http://bknr.net/parenscript
```

or using asdf-install.

```
(asdf-install:install 'parenscript)
```

After downloading the ParenScript sourcecode, set up the ASDF central registry by adding a symlink to "parenscript.asd". Then use ASDF to load ParenScript. You may want to edit the ASDF file to remove the dependency on the Allegroserve HTMLGEN facility.

```
(asdf:oos 'asdf:load-op :parenscript)
```

ParenScript was written by Manuel Odendahl. He can be reached at

```
manuel@bknr.net
```