Using Speakhtml
An Introduction

Niklas Olmes niklas@noxa.de
http://faith.eu.org/

May 18 2003

Abstract

This paper gives a short introduction to Speakhtml, a HTML generator written in Scheme. Speakhtml provides easy to use hooks to make use of Scheme in generating any type of static content with arbitrary complexity. Although using the Scheme programming language gives you great power in designing HTML, it is perfectly possible generating Websites without using or even knowing Scheme. This paper is divided into two sections. The first one serves as a tutorial to Speakhtml. You will need some fundamental knowledge in HTML to understand it. If you don't, perhaps consider learning basic HTML first. The second section is about using Scheme in Speakhtml.

1 Tutorial

This section will show you how a Speakhtml file looks like and how the generator is invocated.

Generally, a Speakhtml file contains a `'('` and a closing `')'`. Further on in this paper we will call any construct with '`(`' at the beginning and '`)`' at the end a list. Note that a file containing only '`()`' would not be valid, since for the Speakhtml parser it contains no data at all. The simplest possible valid Speakhtml file would be a file only containing:

```
(("Hello, World"))
```

As you will have noticed, the data `("Hello World")` is within a list. One can say that a Speakhtml file is one file containing a list of one or more lists.

The given example above would be legal in Speakhtml, but it is not yet a valid HTML page. To make it a valid one, we will have to introduce and use HTML elements in Speakhtml. The generator includes all HTML4 loose elements by default, which are defined in html4-loose.scm. To use a HTML element, prefix the name with '!' and put the result at the beginning of a list with optional contents appended.

```
((!html (!head (!title "Title goes here"))
        (!body ("Hello, World"))))
```

Example above shows a valid HTML "Hello, World" page. We used the `<html>`, `<head>`, and `<body>` tag. Speakhtml will render this to:

```
[..]
<html><head><title>Title goes here</title></head>
<body>Hello, World</body></html>
```

The advantage in using lists instead of the HTML tag notation should be obvious to experienced authors. If not, consider complex table-in-table structures for example.

So how do we tell Speakhtml to generate our "Hello, World"? If you're using the Guile interpreter or any interpreter that is able to give over command line arguments, you may simply do:

```
$ ./speakhtml.scm file-to-parse
```

this would output the rendered result to the console. There is no file option, so redirect output to a file (`> hello.html`).

We haven't talked about strings (text to display) yet. Generally, you have to enclose text in `""` and put it in a list. `("Hello, World")` would be a string in Speakhtml. Inside of tags it is possible to use `""` only, like in `(!b "Hello, World")` or `(!td "Foo!")`. You are allowed to use multiple strings in one tag, of course:

```
("The quick brown" (!i "fox") "jumps over the" (!b "lazy" "dog"))
```

Another aspect we left out until yet are attributes. These are implemented in Speakhtml as vectors. A vector may look like:

```
#(name value)
```

or

```
#((name value))
```

for just one attribut, or:

```
#((name value)
  (name2 value2)
  ...)
```

for multiple attributes. Attributes have to follow directly after the tag name, i.e.:

```
(!body #((text black)
         (link "#0c0c0c"))
   ...)
```

You might have wondered about the vector used in the example above. We did not use string form there. For text, black, and link this is perfectly ok, since Speakhtml automatically converts this labels to strings, as it automatically converts numbers to strings. So why are `""` around `'#0c0c0c'`? This is definately not text, it is a special form. It uses the character `#`, which would be not allowed if not put into string form via `""`. You should be reminded that depending on your Scheme interpreter, case is not expected to be preserved. So you should use explicit string form whenever case matters.

Automatical conversion to string form is included for convenience. Feel free to use it—or not use it. It is up to you.

Complex example:

```
((@d (define author "Your Name your.name@address"))
 (!html (!head (!meta #((http-equiv "Content-Type")
                        (content "text/html; charset=iso-8859-1")))
               (@p `(!meta #((name author)
                             (content ,author))))
               (!title "Put title here"))
        (!body #(text black)
          (!h1 "Title")
          (!div #(align center)
            ("The quick brown fox jumps" (!i "over")
             "the lazy dog")
            (!ul
              (!li "quick")
              (!li "brown")
              (!li "lazy"))))))
```

If you are curious what `@d` and `@p` mean read the next section about the Scheme hooks.

2 Taking Advantage of Speakhtml with Scheme

Don't worry. You actually won't have to program in Scheme. But you may. The last example in the first section shows two special constructs, `@d` and `@p`.

What do they do?

`(@d lis)` evaluates the Scheme list lis

`(@d (define author "blah"))` would give variable author the value "blah".

`(@p lis)` parses the list lis after evaluating it

`(@p '(,author)) would result in parsing ("blah")`

` stands for (quasiquote) and `,` for (unqoute). We used them here to construct a list with variable author expanded. So `,author` evaluates variable author, which results in "blah". In Scheme, there is another qoute, `'` (quote). In `'` `,` has no effect, but it has, if `'` is in a `, since every `,` in a surrounding ` is evaluated.

There are more special forms with `@` prefix in Speakhtml, which are listed here:

`(@d lis)` evaluates the Scheme list lis

`(@e lis)` evaluates the Scheme list lis and displays converted result

`(@p lis)` evaluates the Scheme list lis and parses it afterwards

`(@use "file")`

`(@load "file")` load and execute Scheme file file

`(@input "file")` load Speakhtml file file and parse it

`(@cmt string)` create a comment with string string

We will now take a look at the `@use/@load` directive. It allows you to load a whole Scheme file into the underlying Scheme interpreter. This gives you incredible power in to what you can do with Speakhtml. You may run entire programs within Speakhtml and let the results of the program be parsed by Speakhtml. To allow this, you may call following functions implemented by Speakhtml within Scheme:

`(parse lis)` let Speakhtml parse lis as if it appeared in a Speakhtml file

`(convert x)` try to convert x (whatever it is) into a string

`(comment string)` create a comment with string string

Additionally, Speakhtml provides the constant `FILE`, which is a string containing the name of the current Speakhtml file processed.

Example of a Scheme function for Speakhtml:

```
(define (mypage title date content)
  (parse `(!html (!head (!title ,title))
                 (!body
                   (!h1 ,title)
           ,content
                   (!p)
                   (!b "Last change:" ,date)))))
```

Put above function mypage into a file and include it in your Speakhtml files with `(@use "file")` and use it with `@d`:

```
((@use "mypage.scm")
 (@d (mypage "Title of Page" "Last Date of Modification"
             '("Content of Page"))))
```

As you can see, not even one HTML tag is used in the Speakhtml file. It is perfectly possible to hide HTML by using Speakhtml.

For more elaborated examples in using Speakhtml see sources on http://faith.eu.org/