HtmlPrag Extension

version 0.50

This document was produced using the Skribe Programming Language and its ConTEXt engine.

For further information on Skribe, see `http://www-sop.inria.fr/mimosa/fp/Skribe/`.

Document created on May 26, 2006.

1 Introduction

HtmlPrag is a package developed by Neil Van Dyke. All the hard work was done by Neil. This extension is only a mean to have an easy access to the HtmlPrag. This extension embeds the source of version 0.16 (2005-12-18) of HtmlPrag.

This documentation is mostly copied from the HtmlPrag homepage. Only the exported functions are described here. For more information, please look at the complete HtmlPrag API described in Neil Van Dyke original documentation.

2 HtmlPrag purpose

This text comes from the ***HtmlPrag*** *homepage.*

HtmlPrag provides permissive HTML parsing and emitting capability to Scheme programs. The parser is useful for software agent extraction of information from Web pages, for programmatically transforming HTML files, and for implementing interactive Web browsers. HtmlPrag emits ”SHTML,” which is an encoding of HTML in SXML, so that conventional HTML may be processed with XML tools such as **SXPath**. Like Oleg Kiselyov’s **SSAX-based HTML parser**, HtmlPrag provides a permissive tokenizer, but also attempts to recover structure. HtmlPrag also includes procedures for encoding SHTML in HTML syntax.

The HtmlPrag parsing behavior is permissive in that it accepts erroneous HTML, handling several classes of HTML syntax errors gracefully, without yielding a parse error. This is crucial for parsing arbitrary real-world Web pages, since many pages actually contain syntax errors that would defeat a strict or validating parser. HtmlPrag’s handling of errors is intended to generally emulate popular Web browsers’ interpretation of the structure of erroneous HTML. We euphemistically term this kind of parse ”pragmatic.”

HtmlPrag also has some support for XHTML, although XML namespace qualifiers are currently accepted but stripped from the resulting SHTML. Note that valid XHTML input is of course better handled by a validating XML parser like Kiselyov’s **SSAX**.

3 SHTML and SXML

SHTML is a variant of SXML, with two minor but useful extensions:

1. The SXML keyword symbols, such as `*TOP*`, are defined to be in all uppercase, regardless of the case-sensitivity of the reader of the hosting Scheme implementation in any context. This avoids several pitfalls.

2. Since not all character entity references used in HTML can be converted to Scheme characters in all R5RS Scheme implementations, nor represented in conventional text files or other common external text formats to which one might wish to write SHTML, SHTML adds a special `&` syntax for non-ASCII (or non-Extended-ASCII) characters. The syntax is `(& val)`, where `val` is a symbol or string naming with the symbolic name of the character, or an integer with the numeric value of the character.

```
(make-shtml-entity-value val)
```

STklos
procedure

Yields an SHTML character entity reference for `val`. For example:

```
(make-shtml-entity "rArr")                   ⇒ (& rArr)
(make-shtml-entity (string->symbol "rArr")) ⇒ (& rArr)
(make-shtml-entity 151)                      ⇒ (& 151)
```

`(shtml-entity-value obj)` STklos *procedure*

Yields the value for the SHTML entity `obj`, or `#f` if `obj` is not a recognized entity. Values of named entities are symbols, and values of numeric entities are numbers. An error may raised if `obj` is an entity with system ID inconsistent with its public ID. For example:

```
(define (f s) (shtml-entity-value (cadr (html->shtml s))))
(f "&nbsp;")  ⇒ nbsp
(f "&#2000;") ⇒ 2000
```

4 Tokenizing

The tokenizer is used by the higher-level structural parser, but can also be called directly for debugging purposes or unusual applications. Some of the list structure of tokens, such as for start tag tokens, is mutated and incorporated into the SHTML list structure emitted by the parser.

`(make-html-tokenizer in normalized?)` STklos *procedure*

Constructs an HTML tokenizer procedure on input port `in`. If boolean `normalized?` is true, then tokens will be in a format conducive to use with a parser emitting normalized SXML. Each call to the resulting procedure yields a successive token from the input. When the tokens have been exhausted, the procedure returns the null list. For example:

```
(define input (open-input-string "<a href="foo">bar</a>"))
(define next  (make-html-tokenizer input #f))
(next) ⇒ (a (@ (href "foo")))
(next) ⇒ "bar"
(next) ⇒ (*END* a)
(next) ⇒ ()
(next) ⇒ ()
```

5 Parsing

Most applications will call a parser procedure such as `html->shtml` rather than calling the tokenizer directly.

```
(html->sxml-0nf input)
(html->sxml-1nf input)
(html->sxml-2nf input)
(html->sxml input)
(html->shtml input)
```

STklos *procedure*

Permissively parse HTML from `input`, which is either an input port or a string, and emit an SHTML equivalent or approximation. To borrow and slightly modify an example from Kiselyov's discussion of his HTML parser:

```
(html->shtml
 "<html><head><title></title><title>whatever</title></head><body>
<a href="url">link</a><p align=center><ul compact style="aa">
<p>BLah<!-- comment <comment> --> <i> italic <b> bold <tt> ened</i>
still &lt; bold </b></body><P> But not done yet...")
⇒
(*TOP* (html (head (title) (title "whatever"))
             (body "n"
                   (a (@ (href "url")) "link")
                   (p (@ (align "center"))
                      (ul (@ (compact) (style "aa")) "n"))
                   (p "BLah"
                      (*COMMENT* " comment <comment> ")
                      " "
                      (i " italic " (b " bold " (tt " ened")))
                      "n"
                      "still < bold "))
             (p " But not done yet...")))
```

Note that in the emitted SHTML the text token `"still < bold"` is not inside the `"b"` element, which represents an unfortunate failure to emulate all the quirks-handling behavior of some popular Web browsers.

The procedures `"html->sxml-Xnf"` for `X` 0 through 2 correspond to 0th through 2nd normal forms of SXML as specified in SXML, and indicate the minimal requirements of the emitted SXML.

`html->sxml and` html->shtml `are currently aliases for` html->sxml-0nf`|`.

6 Emiting HTML

```
(write-shtml-as-html shtml)
(write-shtml-as-html shtml out)
(write-shtml-as-html shtml out foreign-filter)
```

STklos *procedure*

Writes a conventional HTML transliteration of the SHTML `shtml` to output port `out`. If `out` is not specified, the default is the current output port. HTML elements of types that are always empty are written using HTML4-compatible XHTML tag syntax.

If `foreign-filter` is specified, it is a procedure of two argument that is applied to any non-SHTML (”foreign”) object encountered in `shtml`, and should yield SHTML. The first argument is the object, and the second argument is a boolean for whether or not the object is part of an attribute value.

No inter-tag whitespace or line breaks not explicit in `shtml` is emitted.

7 Example

The following example shows how one can extract, in a very *ad hoc* way, the titles of the `h1`, `h2` and `h3` sections of the HtmpPrag original HTML documentation.

```
(require "htmlprag")

(define source (open-input-file "../doc/htmlprag-original.html"))

(define (main _)
  (let* ((doc (html->sxml-2nf source))
         (html (cadr doc))
         (head (caddr html))
         (body (cadddr html)))
    (for-each (lambda (x)
                (when (and (pair? x) (member (car x) '(h1 h2 h3)))
                  (format #t "==> ~A\n" (caddr x))))
              body)))
```