

ASXT

Another Scheme XML Transformer

Version 0.1
31 December 2004

Neil W. Van Dyke
neil@neilvandyke.org

<http://www.neilvandyke.org/asxt/>

Copyright © 2004 Neil W. Van Dyke. This program is Free Software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. See the GNU Lesser General Public License [LGPL] for details. For other license options and commercial consulting, contact the author.

1 Introduction

ASXT is another XML transformation library for Scheme.¹ It works with [SXML], including that with extraneous list nesting and nulls, and with [HtmlPrag]’s SHTML variant of SXML.

ASXT is conceptually very simple, and may at some future time be an underlying mechanism in a higher-level language. At this time, ASXT is being used as a practical tool while we attempt to identify what higher-level features and syntax would be most useful for our applications.

Presently, ASXT is somewhat similar to Oleg Kiselyov’s [Pre-post-order] and Kirill Lisovsky’s [STX], and indeed has been informed and inspired by those. ASXT might be described as **pre-post-order** with inheritance and only preorder traversal, and as **stx-engine** without [SXPath]. ASXT’s simplifications were made experimentally, as a starting point for building higher-level features that might be better-suited to some applications. We also expect to incorporate ideas from Jim Bender’s [WebIt].

The ASXT implementation includes a simple “compiler” that produces a closure from input ASXT language. This closure, which may be applied to SXML nodes, should do a fairly efficient job of recursively dispatching on SXML node types to closures, and of assembling the result tree. In most cases, null lists and extraneous list nestings produced by user-supplied closures are discarded quickly and are not included in the output SXML.

The ASXT library requires only R5RS and [SRFI-23].

¹ “ASXT” may be pronounced phonetically, in which case the following mnemonic might be helpful:
“Thrice have I ASXT thee to transform!”

2 Language

The roughly EBNF grammar (in which parentheses represent sexp lists) for the ASXT language is:

```
<action>          ::= <simple-action> | <bindings>

<simple-action>   ::= <proc> | *error* | *null* | *same*

<bindings>      ::= ( [ *inherit* ]
                       [ ( *text*   <simple-action> ) ]
                       [ ( *default* <action>   ) ]
                       { ( <xml-names> <action> ) }* )

<xml-names>     ::= <xml-name> | ( { <xml-name> }+ )
```

The starting nonterminal is *<action>*. An action is something to which an SXML node is applied, and the ASXT starting action is applied to the top-level node of the SXML tree. *<action>* can be a *<simple-action>* or *<bindings>*, but is virtually always the latter for top-level nodes.

<simple-action> can be a closure, which accepts an SXML node, or one of three symbols: **error** causes an error to be signaled, indicating that the node is syntactically invalid; **null** yields the null list and performs no further processing on the node or its children; and **same** yields the node with no further processing.

<bindings> is a list that describes a mapping from SXML elements to actions. When bindings are nested, with a child set of bindings as an action within a parent set of bindings, then the child bindings are applied in that action to the SXML child nodes of the SXML parent element node that triggered the action. This recursive tree traversal is guaranteed to be performed preorder (depth-first), and side-effects are permitted.

Within *<bindings>*, **text** matches SXML text nodes, and *<xml-name>* is a symbol that matches an SXML element of that name. **default** matches any SXML element that is not matched by any given *<xml-name>*. **default** does not match SXML text nodes.

If the first element of a child *<bindings>* is **inherit**, then individual bindings of the immediate parent *<bindings>* (including **default** and any bindings inherited by the parent) are inherited if not overridden in the child *<bindings>*.

If a *<bindings>* does not have **inherit**, then the following bindings are effectively inherited if not overridden:

```
((*text*   *error*)
 (*default* *error*)
 (*COMMENT* *null*))
```

ASXT is normally executed using the `asxt` procedure, which accepts a compiled uncompiled ASXT language and an input SXML. `asxt` may be used within *<proc>* action, such as to interpose additional behavior at a step in an overall ASXT traversal. Note that, since the *<proc>* closure will be applied to the node that was matched for the action, use the `cdr` of that node to descend to child nodes in the traversal, as is shown by the following two functionally equivalent procedures:

```
(asxt '((a ((b ((*text* *same*)
              (c      ((d *same*))))))))
      sxml)

(asxt '((a ((b ,(lambda (node)
                (asxt '((*text* *same*)
                        (c      ((d *same*))))
                        (cdr node))))))
      sxml)
```

For `sxml` input `(a (b (c (d "1"))) (b "2"))`, both expressions yield `((d "1") "2")`.

As an example of using ASXT to extract the links from an HTML input (more precisely, the values of the `href` attributes of `a` elements), using `html->shtml` from [HtmlPrag]:

```
(asxt '((*text*      *null*)
      (*default* (*inherit*))
      (a      (*inherit*
              (@ ((*default* *null*)
                  (href      ((*text* *same*))))))))
      (html->shtml
       "<html><head><title>My Title</title></head><body>
<p>This isn't <a href=\"http://www.yahoo.com/\">Yahoo</a>.</p>
<p><a href=\"http://www.au.org/\">au.org</a> is short.</p>
</body></html>"))
⇒ ("http://www.yahoo.com/" "http://www.au.org/")
```

Naturally, this particular extraction task can be solved much more easily using XPath. The purpose of the example is to illustrate ASXT using a familiar problem. Also, in real-world ASXT programs, good practice would be to yield a list of SXML elements rather than a list of strings, as `asxt` might conceivably concatenate the strings as if they were SXML.

3 API

The ASXT library provides a few Scheme variables and syntax extensions.

<code>asxt-comment-symbol</code>	[Variable]
<code>asxt-entity-symbol</code>	[Variable]
<code>asxt-pi-symbol</code>	[Variable]
<code>asxt-top-symbol</code>	[Variable]

As a convenience for Scheme code that must be portable to implementations with case-sensitivity issues, these variables are bound to the following upper-case symbols used by SXML, respectively: `*COMMENT*`, `*ENTITY*`, `*PI*`, `*TOP*`

<code>asxt</code> <i>action sxml</i>	[Procedure]
--------------------------------------	-------------

Applies ASXT *action* to each of the top level nodes of input *sxml* in turn, and yields the collected result.

<code>compiled-asxt</code> <i>action</i>	[Procedure]
--	-------------

Yields a closure that is the result of compiling ASXT *action*. Note that the `asxt` procedure will automatically compile its ASXT if necessary, so `compiled-asxt` is

mainly useful for avoiding redundant compilations when `asxt` is called many times with the same ASXT. For example:

```
(compiled-asxt '(message ((param ((*default* *same*)))))
⇒ #<procedure>
```

`top-added-asxt` *bindings* [Procedure]

Yields ASXT bindings that are *bindings* with additional bindings added to disregard any SXML **TOP** element that is present. For example:

```
(top-added-asxt '(html (body ...)))
⇒
(html      (body ...))
((@ *PI*) *null*)
(*TOP*    (*inherit*
           (*TOP* *error*)))
```

`asxt-setter` *variable action* [Syntax]

Compiles *action* and defines an ASXT action closure that applies the compiled *action* and both binds *variable* to the result and yields the result.

`asxt-attr-value` *attr-node* [Procedure]

Yields the string value of SXML element attribute *attr-node*. This is used internally by `asxt-attr-setter`, but can also be used directly:

```
(asxt-attr-value '(href ("http://") () ("foo.foo/")))
⇒ "http://foo.foo/"
```

Note that this procedure flattens lists and concatenates strings in the value, although in most cases element attribute values will be represented in SXML as single strings.

`asxt-attr-setter` *variable* [Syntax]

Expands to: `(asxt-setter variable asxt-attr-value)`

For an illustrative example, a program that accepts HTML in SXML form and yields a list of the URLs of `img` elements along with any width and height information specified for each in the HTML:

```
(define scraped-imgs
  (compiled-asxt
    '((*text* *null*)
      (*default* (*inherit*))
      (@ *null*)
      (img      ,(lambda (node)
                  (let ((s #f) (w #f) (h #f))
                    (asxt '((@ ((*default* *null*)
                                (src      ,(asxt-attr-setter s))
                                (width   ,(asxt-attr-setter w))
                                (height  ,(asxt-attr-setter h))))
                          (cdr node))
                    (vector s w h)))))))
  (asxt scraped-imgs
    '(html (body (img (@ (height "60") (src "b.jpg") (width "80")))
                    (p "Stage Right: "
                      (img (@ (src "a.jpg") (align "right"))))))))
  ⇒ (#("b.jpg" "80" "60") #("a.jpg" #f #f))
```

References

- [HtmlPrag] Neil W. Van Dyke, “HtmlPrag: Pragmatic Parsing of HTML to SHTML and SXML.”
<http://www.neilvandyke.org/htmlprag/>
- [LGPL] Free Software Foundation, “GNU Lesser General Public License,” Version 2.1, February 1999, 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
<http://www.gnu.org/copyleft/lesser.html>
- [Pre-post-order] Oleg Kiselyov, “XML/HTML processing in Scheme: SXML expression tree transformers,”
<http://pobox.com/~oleg/ftp/Scheme/lib/SXML-tree-trans.scm>
- [SRFI-23] Stephan Houben, “Error reporting mechanism,” SRFI 23, 26 April 2001.
<http://srfi.schemers.org/srfi-23/srfi-23.html>
- [SSAX] Oleg Kiselyov, “A functional-style framework to parse XML documents,” 5 September 2002.
<http://pobox.com/~oleg/ftp/Scheme/xml.html#XML-parser>
- [STX] Kirill Lisovsky, “STX,”
<http://www.pair.com/lisovsky/transform/stx/>
- [SXML] Oleg Kiselyov, “SXML,” revision 3.0.
<http://pobox.com/~oleg/ftp/Scheme/SXML.html>
- [SXPath] Kirill Lisovsky, “SXPath - SXML Query Language,”
<http://pair.com/lisovsky/query/sxpath/>
- [WebIt] Jim Bender, “WebIt! - An XML Framework for Scheme.”
<http://celtic.benderweb.net/webit/>