previous next contents top

SXML

This page specifies **revision 3.0** of SXML. SXML is an abstract syntax tree of an XML document. SXML is also a concrete representation of the XML Infoset in the form of S-expressions. The generic tree structure of SXML lends itself to a compact library of combinators for querying and transforming SXML.

> The master SXML specification file is written in SXML itself. The present web page is the result of translating that SXML code with the appropriate "stylesheet". The master file, its renditions in HTML and other formats, and the corresponding stylesheets are available at
> <xml.html>.

Introduction

An XML document is essentially a tree structure. The start and the end tags of the root element enclose the document content, which may include other elements or arbitrary character data. Text with familiar angular brackets is an external representation of an XML document. Applications ought to deal with an internalized form: an XML Information Set, or its specializations. This form lets an application locate specific data or transform an XML tree into another tree, which can then be written out as an XML, HTML, PDF, etc. document.

XML Information Set (Infoset) [XML Infoset] is an abstract datatype that describes information available in a well-formed XML document. Infoset is made of "information items", which denote elements, attributes, character data, processing instructions, and other components of the document. Each information item has a number of associated properties, e.g., name, namespace URI. Some properties -- for example, 'children' and 'attributes' -- are collections of other information items. Infoset describes only the information in an XML document that is relevant to applications. The default value of attributes declared in the DTD, parameter entities, the order of attributes within a start-tag, and other data used merely for parsing or validation are not included. Although technically Infoset is specified for XML it largely applies to other semi-structured data formats, in particular, HTML.

The hierarchy of containers comprised of text strings and other containers greatly lends itself to be described by S-expressions. S-expressions [McCarthy] are easy to parse into an internal representation suitable for traversal. They have a simple external notation (albeit with many a parentheses), which is relatively easy to compose even by hand. S-expressions have another advantage: *provided* an appropriate design, they can represent Scheme code to be evaluated. This code-data dualism is a distinguished feature of Lisp and Scheme.

SXML is a concrete instance of the XML Infoset. Infoset's goal is to present in some form all relevant pieces of data and their *abstract*, container-slot relationships with each other. SXML gives the nest of containers a concrete realization as S-expressions, and provides means of accessing items and their properties. SXML is a "relative" of XPath [XPath] and DOM [DOM], whose data models are two other instances of the XML Infoset. SXML is particularly suitable for

Scheme-based XML/HTML authoring, SXPath queries, and tree transformations. In John Hughes' terminology [Hughes-PP], SXML is a term implementation of evaluation of the XML document.

Notation

We will use an Extended BNF Notation (EBNF) employed in the XML Recommendation [XML]. The following table summarizes the notation.

thing`?`
: An optional *thing*

thing`*`
: Zero or more *thing*s

thing`+`
: One or more *thing*s

thing1 `|` thing2 `|` thing3
: Choice of *thing*s

`<thing>`
: A non-terminal of a grammar

`thing`
: A terminal of the grammar that is a Scheme identifier

`"thing"`
: A terminal of the grammar that is a Scheme string

`thing`
: A literal Scheme symbol

`(` `<A> <B>* <C>?` **`)`**
: An S-expression made of `<A>` followed by zero or more `<B>` and, afterwards, optionally by `<C>`

`{`*`A`* `<B>*` **`}`**
: A tagged set: an S-expression made of *`A`* followed by zero or more instances of `<B>` in any order

`(` `<A> . <B>` **`)`**
: An S-expression that is made by prepending `<A>` to an S-expression denoted by `<B>`

***`make-symbol``(`***`<A>:<B>`**`)`**
: A symbol whose string representation consists of all characters that spell `<A>` followed by the colon character and by the characters that spell `<B>`. The ***`make-symbol()`*** notation can be regarded a meta-function that creates symbols.

Grammar

[1] `<TOP>` `::=` **`(`** *`*TOP*`* `<annotations>? <PI>* <comment>* <Element>` **`)`**

This S-expression stands for the root of the SXML tree, a document information item of the Infoset. Its only child element is the root element of the XML document.

[2] `<Element>` `::=` **`(`** `<name> <annot-attributes>? <child-of-element>*` **`)`**

[3] `<annot-attributes>` `::=` **`{`***`@`* `<attribute>* <annotations>?` **`}`**

[4] `<attribute>` `::=` **`(`** `<name> "value"? <annotations>?` **`)`**

[5] `<child-of-element>` `::=` `<Element> | "character data" | <PI> | <comment> | <entity>`

These are the basic constructs of SXML.

[6] `<PI>` `::=` **`(`** *`*PI*`* `pi-target <annotations>? "processing instruction content string"` **`)`**

The XML Recommendation specifies that processing instructions (PI) are distinct from elements and character data; processing instructions must be passed through to applications. In SXML, PIs are therefore represented by nodes of a dedicated type `*PI*`. DOM Level 2 treats processing instructions in a similar way.

```
[7] <comment> ::= ( *COMMENT* "comment string" )
[8]  <entity> ::= ( *ENTITY* "public-id" "system-id" )
```

Comments are mentioned for completeness. A SSAX XML parser [SSAX], among others, transparently skips the comments. The XML Recommendation permits the parser to pass the comments to an application or to completely disregard them. The present SXML grammar admits comment nodes but does not mandate them by any means.
An `<entity>` node represents a reference to an unexpanded external entity. This node corresponds to an unexpanded entity reference information item, defined in Section 2.5 of [XML Infoset]. Internal parsed entities are always expanded by the XML processor at the point of their reference in the body of the document.

```
 [9]            <name> ::= <LocalName> | <ExpName>
[10]       <LocalName> ::= NCName
[11]         <ExpName> ::= make-symbol(<namespace-id>:<LocalName>)
[12]    <namespace-id> ::= make-symbol("URI") | user-ns-shortcut
[13]      <namespaces> ::= {*NAMESPACES* <namespace-assoc>* }
[14] <namespace-assoc> ::= ( <namespace-id> "URI" original-prefix? )
```

An SXML `<name>` is a single symbol. It is generally an expanded name [XML-Namespaces], which conceptually consists of a local name and a namespace URI. The latter part may be empty, in which case `<name>` is a `NCName`: a Scheme symbol whose spelling conforms to production [4] of the XML Namespaces Recommendation [XML-Namespaces]. `<ExpName>` is also a Scheme symbol, whose string representation contains an embedded colon that joins the local and the namespace parts of the name. A **`make-symbol("URI")`** is a Namespace URI string converted to a Scheme symbol. Universal Resource Identifiers (URI) may contain characters (e.g., parentheses) that are prohibited in Scheme identifiers. Such characters must be %-quoted during the conversion from a URI string to `<namespace-id>`. The original XML Namespace prefix of a QName [XML-Namespaces] may be retained as an optional member *`original-prefix`* of a `<namespace-assoc>` association. A `user-ns-shortcut` is a Scheme symbol chosen by an application programmer to represent a namespace URI in the application program. The SSAX parser lets the programmer define (short and mnemonic) unique shortcuts for often long and unwieldy Namespace URIs.

Annotations

```
[15] <annotations> ::= {@ <namespaces>? <annotation>* }
[16]  <annotation> ::= To be defined in the future
```

The XML Recommendation and related standards are not firmly fixed, as the long list of errata and version 1.1 of XML clearly show. Therefore, SXML has to be able to accommodate future changes while guaranteeing backwards compatibility. SXML also ought to permit applications to store various processing information (e.g., cached resolved IDREFs) in an SXML tree. A hash of ID-type attributes would, for instance, let us implement efficient lookups in (SOAP-) encoded arrays. To allow such extensibility, we introduce two new node types: `<annotations>` and `<annotation>`. The semantics of the latter is to be established in future versions of SXML. Possible examples of an `<annotation>` are the unique id of an element or the reference to element's parent.

The structure and the semantics of `<annotations>` is similar to those of an attribute list. In a manner of speaking, annotations are ``attributes'' of an attribute list. The tag `@` marks a collection of ancillary data associated with an SXML node. For an element SXML node, the ancillary collection is that of attributes. A nested `@` list is therefore a collection of ``second-level'' attributes -- annotations -- such as namespace nodes, parent pointers, etc. This design seems to be in accord with the spirit of the XML Recommendation, which uses XML attributes for two distinct purposes. Genuine, semantic attributes provide ancillary description of the corresponding XML element, e.g.,

```
<weight units='kg'>16</weight>
```

On the other hand, attributes such as `xmlns`, `xml:prefix`, `xml:lang` and `xml:space` are auxiliary, or being used by XML itself. The XML Recommendation distinguishes auxiliary attributes by their prefix `xml`. SXML groups all such auxiliary attributes into a `@`-tagged list inside the attribute list.

XML attributes are treated as a dust bin. For example, the XSLT Recommendation allows extra attributes in `xslt:template`, provided these attributes are in a non-XSLT namespace. A user may therefore annotate an XSLT template with his own attributes, which will be silently disregarded by an XSLT processor because the processor never looks for them. RELAX/NG explicitly lets a schema author specify that an element may have more attributes than given in the schema, provided those attributes come from a particular namespace. The presence of these extra attributes should not affect the XML processing applications that do not specifically look for them. Annotations such as parent pointers and the source location information are similarly targeted at specific applications. The other applications should not be affected by the presence or absence of annotations. Placing the collection of annotations inside the attribute list accomplishes that goal.

Annotations can be assigned to an element and to an attribute of an element. The following example illustrates the difference between the two annotations, which, in the example, contain only one annotation: a pointer to the parent of a node.

```
(a (@
     (href "http://somewhere/"
       (@ (*parent* a-node))       ; <annotations> of the attribute 'href'
       )
     (@ (*parent* a-parent-node))) ; <annotations> of the element 'a'
   "link")
```

The `<TOP>` node may also contain annotations: for example, `<namespaces>` for the entire document or an index of ID-type attributes.

```
(*TOP* (@ (id-collection id-hash)) (p (@ (id "id1")) "par1"))
```

Annotations of the `<TOP>` element look exactly like `attributes' of the element. That should not cause any confusion because `<TOP>` cannot have genuine attributes. The SXML element `<TOP>` is an abstract representation of the whole document and does not correspond to any single XML element. Assigning annotations, which look and feel like an attribute list, to the `<TOP>` element does not contradict the Infoset Recommendation, which specifically states that it is not intended to be exhaustive. Attributes in general are not considered children of their parents, therefore, even with our annotations the `<TOP>` element has only one child -- the root element.

SXML Tree

Infoset's information item is a sum of its properties. This makes a list a particularly suitable data structure to represent an item. The head of the list, a Scheme identifier, *names* the item. For many items this is their (expanded) name. For an information item that denotes an XML element, the corresponding list starts with element's expanded name, optionally followed by a collection of attributes and annotations. The rest of the element item list is an ordered sequence of element's children -- character data, processing instructions, and other elements. Every child is unique; items never share their children even if the latter have the identical content.

A `parent` property of an Infoset information item might seem troublesome. The Infoset Recommendation [XML Infoset] specifies that element, attribute and other kinds of information items have a property `parent`, whose value is an information item that contains the given item in its `children` property. The property parent thus is an upward link from a child to its parent. At first sight, S-expressions seem lacking in that respect: S-expressions represent directed trees and trees cannot have upward links. An article [Parent-pointers] discusses and compares five different methods of determining the parent of an SXML node. The existence of these methods is a crucial step in a constructive proof that SXML is a complete model of the XML Information set and the SXML query language (SXPath) can fully implement the XPath Recommendation.

Just as XPath does and the Infoset specification explicitly allows, we group character information items into maximal text strings. The value of an attribute is normally a string; it may be omitted (in case of HTML) for a boolean attribute, e.g., `<option checked>`.

We consider a collection of attributes an information item in its own right, tagged with a special name `@`. The character '@' may not occur in a valid XML name; therefore `<annot-attributes>` cannot be mistaken for a list that represents an element. An XML document renders attributes, processing instructions, namespace specifications and other meta-data differently from the element markup. In contrast, SXML represents element content and meta-data uniformly -- as tagged lists. RELAX-NG also aims to treat attributes as uniformly as possible with elements. This uniform treatment, argues James Clark [RNG-Design], is a significant factor in simplifying the language. SXML takes advantage of the fact that every XML name is also a valid Scheme identifier, but not every Scheme identifier is a valid XML name. This observation lets us introduce administrative names such as `@`, `*PI*`, `*NAMESPACES*` without worrying about potential name clashes. The observation also makes the relationship between XML and SXML well-defined. An XML document converted to SXML can be reconstructed into an equivalent (in terms of the Infoset) XML document. Moreover, due to the implementation freedom given by the Infoset specification, SXML itself is an instance of the Infoset.

Since an SXML document is essentially a tree structure, the SXML grammar of Section 3 can be presented in the following, more uniform form:

```
[N] <Node> ::= <Element> | <annot-attributes> | <attribute> | "character data: text string" |
               <namespaces> | <TOP> | <PI> | <comment> | <entity> | <annotations> | <annotation>
```

or as a set of two mutually-recursive datatypes, `Node` and `Nodelist`, where the latter is a list of `Node`s:

```
[N1]     <Node> ::= ( <name> . <Nodelist> ) | "text string"
[N2] <Nodelist> ::= ( <Node>* )
[N3]     <name> ::= <LocalName> | <ExpName> | @ | *TOP* | *PI* | *COMMENT* | *ENTITY* |
                    *NAMESPACES*
```

The uniformity of the SXML representation for elements, attributes, and processing instructions simplifies queries and transformations. In our formulation, attributes and processing instructions look like regular elements with a distinguished name. Therefore, query and transformation functions dedicated to attributes become redundant.

A function `SSAX:XML->SXML` of a functional Scheme XML parsing framework SSAX [SSAX] can convert an XML document or a well-formed part of it into the corresponding SXML form. The parser supports namespaces, character and parsed entities, attribute value normalization, processing instructions and CDATA sections.

Namespaces

The motivation for XML Namespaces is explained in an excellent article by James Clark [Clark1999]. He says in part:

> The XML Namespaces Recommendation tries to improve this situation by extending the data model to allow element type names and attribute names to be qualified with a URI. Thus a document that describes parts of cars can use `part` qualified by one URI; and a document that describes parts of books can use `part` qualified by another URI. I'll call the combination of a local name and a qualifying URI a universal name. The role of the URI in a universal name is purely to allow applications to recognize the name. There are no guarantees about the resource identified by the URI. The XML Namespaces Recommendation does not require element type names and attribute names to be universal names; they are also allowed to be local names.
> ...
> The XML Namespaces Recommendation expresses universal names in an indirect way that is compatible with XML 1.0. In effect the XML Namespaces Recommendation defines a mapping from an XML 1.0 tree where element type names and attribute names are local names into a tree where element type names and attribute names can be universal names. The mapping is based on the idea of a prefix. If an element type

name or attribute name contains a colon, then the mapping treats the part of the name before the colon as a prefix, and the part of the name after the colon as the local name. A prefix `foo` refers to the URI specified in the value of the `xmlns:foo` attribute. So, for example

```
<cars:part xmlns:cars='http://www.cars.com/xml'/>
```

maps to

```
<{http://www.cars.com/xml}part/>
```

Note that the `xmlns:cars` attribute has been removed by the mapping.

Using James Clark's terminology, SXML as defined by [N1] is precisely that tree where element type names and attribute names can be universal names. According to productions [N3] and [9-12], a universal name, `<name>`, is either a local name or an expanded name. Both kinds of names are Scheme identifiers. A local name has no colon characters in its spelling. An expanded name is spelled with at least one colon, which may make the identifier look rather odd. In SXML, James Clark's example will appear as follows:

```
(http://www.cars.com/xml:part)
```

or, somewhat redundantly,

```
(http://www.cars.com/xml:part
  (@ (@ (*NAMESPACES* (http://www.cars.com/xml "http://www.cars.com/xml" cars)))))
```

Such a representation also agrees with the Namespaces Recommendation [XML-Namespaces], which says: "Note that the prefix functions only as a placeholder for a namespace name. Applications should use the namespace name, not the prefix, in constructing names whose scope extends beyond the containing document."

It may be unwieldy to deal with identifiers such as `http://www.cars.com/xml:part`. Therefore, an application that invokes the SSAX parser may tell the parser to map the URI `http://www.cars.com/xml` to an application-specific namespace shortcut `user-ns-shortcut`, e.g., `c`. The parser will then produce

```
(c:part (@ (@ (*NAMESPACES* (c "http://www.cars.com/xml")))))
```

To be more precise, the parser will return just

```
(c:part)
```

If an application told the parser how to map `http://www.cars.com/xml`, the application can keep this mapping in its mind and will not need additional reminders.

On the other hand, we should not be afraid of SXML node names such as `http://www.cars.com/xml:part`. These names are Scheme symbols. No matter how long the name of a symbol may be, it is fully spelled only once, in the symbol table. All other occurrences of the symbol are short references to the corresponding slot in the symbol table.

We must note a 1-to-1 correspondence between `user-ns-shortcuts` and the corresponding namespace URIs. This is generally not true for XML namespace prefixes and namespace URIs. A `user-ns-shortcut` uniquely represents the corresponding namespace URI within the document, but an XML namespace prefix does not. For example, different XML prefixes may specify the same namespace URI; XML namespace prefixes may be redefined in children elements. The other difference between `user-ns-shortcuts` and XML namespace prefixes is that the latter are at the whims of the author of the document whereas the namespace shortcuts are defined by an XML processing application. The shortcuts are syntactic sugar for namespace URIs.

The list of associations between namespace IDs and namespace URIs (and, optionally, original XML Namespace prefixes) is an optional member of an `<annotations>`. For regular elements, `<namespaces>` will contain only those namespace declarations that are relevant for that element. Most of the time however `<namespaces>` in the `<annotations>` will be absent.

The node `<namespaces>`, if present as an annotation of the `<TOP>` element, must contain `<namespace-assoc>` for the whole document. It may happen that one namespace URI is associated in the source XML document with several namespace prefixes. There will be then several corresponding `<namespace-assoc>` differing only in the *`original-prefix`* part.

The topic of namespaces in (S)XML and (S)XPath is thoroughly discussed in [Lisovsky-NS] and [SXML-NS].

Case-sensitivity of SXML names

XML is a case-sensitive language. The names of XML elements, attributes, targets of processing instructions, etc. may be distinguished solely by their case. These names however are represented as Scheme identifiers in SXML. Although Scheme is traditionally a case-insensitive language, the use of Scheme symbols for XML names poses no contradictions. According to R5RS [R5RS],

> (symbol->string symbol) returns the name of symbol as a string. If the symbol was part of an object returned as the value of a literal expression (section 4.1.2) or by a call to the read procedure, and its name contains alphabetic characters, then the string returned will contain characters in the implementation's preferred standard case -- some implementations will prefer upper case, others lower case. If the symbol was returned by string->symbol, the case of characters in the string returned will be the same as the case in the string that was passed to string->symbol.

Thus, R5RS explicitly permits case-sensitive symbols: `(string->symbol "a")` is always different from `(string->symbol "A")`. SXML uses such case-sensitive symbols for all XML names. SXML-compliant XML parsers must preserve the case of all names when converting them into symbols. A parser may use the R5RS procedure `string->symbol` or other available means.

Reading and writing SXML documents may require special care. The cited R5RS section allows a Scheme system to alter the case of symbols found in a literal expression or read from input ports. Entering SXML names on case-insensitive systems requires the use of a bar notation, `string->symbol` or other standard or non-standard ways of producing case-sensitive symbols. A SSAX built-in test suite is a highly portable example of entering literal case-sensitive SXML names. Such workarounds, however simple, become unnecessary on Scheme systems that support DSSSL. According to the DSSSL standard,

> 7.3.1. Case Sensitivity. Upper- and lower-case forms of a letter are always distinguished. NOTE 5: Traditionally Lisp systems are case-insensitive.

In addition, a great number of Scheme systems offer a case-sensitive reader, which often has to be activated through a compiler option or pragma. A web page [Scheme-case-sensitivity] discusses case sensitivity of various Scheme systems in detail.

Normalized SXML

Normalized SXML is a proper subset of SXML optimized for efficient processing. An SXML document in a normalized form must satisfy a number of additional restrictions. Most of the restrictions concern the order and the appearance of the `<annot-attributes>` and `<annotations>` within an `<Element>` node, productions [2-4]. In the spirit of the relational database model, we introduce a number of increasingly rigorous normal forms for SXML expressions. An SXML document in normal form N is also in normal form M for any M<N. The higher normal forms impose more constraints on the structure of SXML expressions but in return permit faster access.

The most permissive 0NF may be considered a relaxation of the SXML grammar. The form does not mandate the relative order of `<annot-attributes>`. The attribute list, if present, may be inter-mixed with `<child-of-element>`. SGML provides two equal forms for boolean attributes: minimized, e.g., `<OPTION checked>` and full, `<OPTION checked="checked">`. XML mandates the full form only, whereas HTML allows both, favoring the former. 0NF SXML supports the minimized form along with the full one: `(OPTION (@ (checked)))` and `(OPTION (@ (checked "checked")))`.

In the 1NF, optional `<annot-attributes>` if present must precede all other components of an `<Element>` SXML node. This is the order reflected in Production [2]. Boolean attributes must appear in their full form, e.g., `(OPTION (@ (checked "checked")))`. The 1NF is the "recommended" normal form.

The 2NF makes `<annot-attributes>` a required component of an SXML element node. If an element has no attributes nor annotations, `<annot-attributes>` shall be specified as `(@)`. In the 2NF, `<comment>` and `<entity>` nodes must be absent. Parsed entities should be expanded, even if they are external.

In the third normal form 3NF all text strings must be joined into maximal text strings: no `<Nodelist>` shall contain two adjacent text-string nodes.

The normal forms make it possible to access SXML items in efficient ways. If an SXML document is known to be in the 3NF, for example, an application never has to check for the existence of `<annot-attributes>`. Checking for child nodes and retrieving text data are simplified as well.

Examples

Simple examples:

```
(some-name)        ; An empty element without attributes (in 0NF and 1NF)
(some-name (@))    ; The same but in the normalized (2NF) form
```

The figure below shows progressively more complex examples.

XML	SXML

`<WEIGHT unit="pound">`
`  <NET certified="certified">67</NET>`
`  <GROSS>95</GROSS>`
`</WEIGHT>`
	`(WEIGHT (@ (unit "pound"))`
	`  (NET (@ (certified)) 67)`
	`  (GROSS 95)`
	`)`

` `	`(BR)`
` </BR>`	`(BR)`
`<P>` `<![CDATA[ \r\n` `<![CDATA[ ]]]]>> </P>`	`(*TOP*` `  (P " \n<![CDATA[ ]]> "))`
An example from the XML Namespaces Recommendation	

`<!-- initially, the default`
`  namespace is 'books' -->`
`<book xmlns='urn:loc.gov:books'`
`  xmlns:isbn='urn:ISBN:0-395-36341-6'>`
`  <title>Cheaper by the Dozen</title>`
`  <isbn:number>1568491379</isbn:number>`
`  <notes>`
`  <!-- make HTML the default namespace`
`  for some commentary -->`
`  <p xmlns='urn:w3-org-ns:HTML'>`
`  This is a <i>funny</i> book!`
`  </p>`
`  </notes>`
`</book>`
	`(*TOP*`
	`  (urn:loc.gov:books:book`
	`  (urn:loc.gov:books:title`
	`  "Cheaper by the Dozen")`
	`  (urn:ISBN:0-395-36341-6:number "1568491379")`
	`  (urn:loc.gov:books:notes`
	`  (urn:w3-org-ns:HTML:p`
	`  "This is a "`
	`  (urn:w3-org-ns:HTML:i "funny")`
	`  " book!"))))`

Another example from the XML Namespaces Recommendation

```
<RESERVATION
  xmlns:HTML=
   'http://www.w3.org/TR/REC-html40'>
<NAME HTML:CLASS="largeSansSerif">
    Layman, A</NAME>
<SEAT CLASS='Y'
  HTML:CLASS="largeMonotype">33B</SEAT>
<HTML:A HREF='/cgi-bin/ResStatus'>
    Check Status</HTML:A>
<DEPARTURE>1997-05-24T07:55:00+1
</DEPARTURE></RESERVATION>
```

```
(*TOP*
  (@ (*NAMESPACES*
       (HTML "http://www.w3.org/TR/REC-html40")))
  (RESERVATION
    (NAME (@ (HTML:CLASS "largeSansSerif"))
      "Layman, A")
    (SEAT (@ (HTML:CLASS "largeMonotype")
             (CLASS "Y"))
      "33B")
    (HTML:A (@ (HREF "/cgi-bin/ResStatus"))
      "Check Status")
    (DEPARTURE "1997-05-24T07:55:00+1")))
```

Acknowledgment

I am indebted to Kirill Lisovsky for numerous discussions and suggestions. He shares the credit for this page. The errors are all mine.

References

[McCarthy] John McCarthy. Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I. Comm. ACM, 3(4):184-195, April 1960.
<http://www-formal.stanford.edu/jmc/recursive/recursive.html>

[Clark1999] Jim Clark. XML Namespaces. February 4, 1999.
<http://www.jclark.com/xml/xmlns.htm>

[Clark2001] James Clark, The Design of RELAX NG. December 6, 2001.
<http://www.thaiopensource.com/relaxng/design.html>

[Hughes1995] John Hughes, The Design of a Pretty-printing Library. Advanced Functional Programming, J. Jeuring and E. Meijer, Eds. Springer Verlag, LNCS 925, 1995, pp. 53-96.
<http://www.cs.chalmers.se/~rjmh/Papers/pretty.html>

[R5RS] R. Kelsey, W. Clinger, J. Rees (eds.), Revised5 Report on the Algorithmic Language Scheme. Higher-Order and Symbolic Computation, Vol. 11, No. 1, September, 1998 and ACM SIGPLAN Notices, Vol. 33, No. 9, October, 1998.
<http://www.schemers.org/Documents/Standards/R5RS/>

[SSAX] Oleg Kiselyov. Functional XML parsing framework: SAX/DOM and SXML parsers with support for XML Namespaces and validation. September 5, 2001.
<xml.html#XML-parser>

[SXML-short-paper] Oleg Kiselyov. XML and Scheme. An introduction to SXML and SXPath; illustration of SXPath expressiveness and comparison with XPath. September 17, 2000.
<Scheme/SXML-short-paper.html>

[Parent-pointers] Oleg Kiselyov. On parent pointers in SXML trees. February 12, 2003.
<Scheme/xml.html#parent-ptr>

[Lisovsky] Kirill Lisovsky. Case sensitivity of Scheme systems.
<http://pair.com/lisovsky/scheme/casesens/index.html>

[Lisovsky-NS] Kirill Lisovsky. Namespaces in XML and SXML.
<http://pair.com/lisovsky/xml/ns/>

[SXML-NS] Namespaces in SXML and (S)XPath. Discussion thread on the SSAX-SXML mailing list. May 28, 2002 and June 7, 2002.
<http://sourceforge.net/mailarchive/forum.php?thread_id=759249&forum_id=599>
<http://sourceforge.net/mailarchive/forum.php?thread_id=789156&forum_id=599>

[Annotations] An alternative syntax for aux-list. Discussion thread on the SSAX-SXML mailing list. January 5-14, 2004.
<http://article.gmane.org/gmane.lisp.scheme.ssax-sxml/37>

[DOM] World Wide Web Consortium. Document Object Model (DOM) Level 1 Specification. W3C Recommendation.
<http://www.w3.org/TR/REC-DOM-Level-1>

[XML] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation. October 6, 2000.
<http://www.w3.org/TR/REC-xml>

[XML Infoset] World Wide Web Consortium. XML Information Set. W3C Recommendation. 24 October 2001.
<http://www.w3.org/TR/xml-infoset>

[XML-Namespaces] World Wide Web Consortium. Namespaces in XML. W3C Recommendation. January 14, 1999.
<http://www.w3.org/TR/REC-xml-names/>

[XPath] World Wide Web Consortium. XML Path Language (XPath). Version 1.0. W3C Recommendation. November 16, 1999.
<http://www.w3.org/TR/xpath>

Changes from the previous version

Following the suggestion by Matthias Radestock, `<aux-list>` is renamed into `<annotations>`, and `<aux-node>` into `<annotation>`. What used to be called `<attribute-list>` is now `<annot-attributes>`.

Jim Bender suggested annotations in `<PI>` nodes, for example, to record the location of the processing instruction in the document source.

We have added a detailed discussion of annotations on a single attribute, and of `<TOP>` annotations.

We have introduced the notation for a tagged set **{***a* `<B> <C>` **}** to precisely specify the syntax of `<annot-attributes>` and `<annotations>`.

Previously SXML defined an attributes list and an aux-list as:

```
[3]  <attributes-list> ::= ( @ <attribute>* )
[15] <aux-list> ::= ( @@ <namespaces>? <aux-node>* )
```

Both lists looked alike, as tagged associative lists. Attribute lists were tagged with a distinguished symbol `@` and aux-lists were tagged with a distinguished symbol `@@`. Both lists were ``improper" children of their parent SXML element. Both lists were optional. An aux-list contained `auxiliary' associations, e.g., the information about original namespace prefixes or the pointer to the parent SXML element. Here is an example of an SXML element with both attributes-list and aux-list:

```
(tag (@ (attr "val")) (@@ (*parent* val)) kid1 kid2)
```

In a normalized SXML (2NF), both lists had to be present, and appear in the right order among the children of an element. The empty attributes-list had to be coded as `(@)` and the empty aux-list had to be coded as `(@@)`.

In the present version, `<attributes-list>` is renamed into `<annot-attributes>`, and `<aux-list>` into `<annotations>`. Both `<annot-attributes>` and `<annotations>` are tagged with the same symbol: `@`. Annotations may no longer appear among the children of an element. Rather, annotations may only appear inside `<annot-attributes>` and at the top level. The previous example reads now as follows:

```
(tag (@ (attr "val") (@ (*parent* val))) kid1 kid2)
```

The new format for annotations makes it easier to skip them when they are not needed. If an SXML application only looks up attributes by their names, the change in syntax is transparent. The transparency of annotation nodes in SXPath is one reason for using the same symbol `@` to tag both `<annot-attributes>` and `<annotations>`. In more detail, this point is discussed in [Annotations]. It seems that most of the existing SXML processing code will not be affected by the change. Placing annotations inside `<annot-attributes>` seems to make it easier to process them during SXSLT traversals. In fact, that was the original motivation for the change in syntax.

The new format makes SXML more space efficient, for documents where most elements have no annotations nor attributes. Indeed, an SXML node without attributes and annotations previously had the following 3NF form:

```
(tag (@) (@@) data)
```

Now, the same node is realized as `(tag (@) data)`

A detailed discussion is given in [Annotations].

Previously, annotations were mandatory in 3NF. If a node had no annotations, its aux-list had to be specified as `(@@)`. The aux-list had to appear at the fixed position, as the third member of an SXML element node, so that we could easily locate the proper children of a node without extra tests. In the present version of SXML, `<annotations>` are included in `<annot-attributes>`. Attribute lists are considered unordered and are typically handled with the help of `assq`. Specifying a fixed position for annotations or making them mandatory in the 3NF no longer seems reasonable nor would it noticeably speed up SXML processing. Thus, if an element has no annotations, `<annotations>` is absent. If the element also has no attributes, the empty `annot-attributes` list `(@)` must still be present, in 2NF and higher.

Last updated March 12, 2004

This site's top page is **http://okmij.org/ftp/**

oleg-at-okmij.org
Your comments, problem reports, questions are very welcome!