Using Semi-xml

Marcel Timmerman

Abstract

The Semi-xml package comprises of a set of Perl 6 modules to convert a language coined 'semi xml' or sxml text into *XML* languages such as *HTML*, *SOAP* or *XSL*.

In the package there is also a program called sxml2xml which uses this library to transform sxml text from a file into XML after which it is stored in another file. The generated XML can also be send to any program for conversion to other formats or for checking. Examples are xsltproc, xmllint, rnv, wkhtmltopdf, xep etcetera.

The latest version of this document is generated on date 2015-08-27

Table of Contents

1. Introduction of module Semi-xml

Welcome to *Semi-xml*. As mentioned in the abstract, the modules in the library helps the programmer to convert sxml text into *XML*.

1.1. Advantages using this language

- It is more clear to read the text which is where you want to focus on and therefor more maintainable than XML is. One of those thing I find annoying is writing the end tag. This is solved by using '[' and ']' to enclose a deeper level of nesting.

- Some language constructs will help inserting predefined text. An example of this might be to insert a lot of attributes of some element and hide it under a name.

- There are also constructs which are able to call code in modules to simplify things. An example is the use of one of the core methods to insert a date or time-stamp in the text. As an example above

at the end of the abstract a date is generated. This date will always be the current date when the text is processed again. This specific part adds dynamism to your text.

- The language is extensible. Developers can add their own libraries to load for example data from a database and inserting it in the text as a HTML table.

1.2. Disadvantages

- There is an extra level of processing. If the XML text is simple one should not be bothered learning this language. Especially when the dynamic constructs aren't used.
- This project is only just started. There are many things left on the wish list. E.g. I would like to give proper messages when mistakes are made in the syntax.

1.3. Information given in this article

- *The language*. Here the basic layout will be explained, the terminology and meaning of the syntax. The start will be gentle showing simple cases first..
  - *The document section*. The document is where text is written for the specific documents together with the elements to format the text. Some of the special elements will need the prelude section explained in the next section.
  - *The prelude section*. The prelude section will be explained. In this section you can control the output of the result. Some options are only used by the program sxml2xml. Defaults are also explained.
- *The programmers view*. Developers can write new modules with methods to process tasks not yet captured by this package.
  - *Classes data and methods*. An explanation of the layout of a users module and an explanation of how to specify the options in the prelude
  - *Substitution data*. Declarations of data which can be used to substitute into the text.
  - *Methods*. Methods which help inserting new text and entities into the text.
- *Using the program*. The program is used to read sxml from a file and saved or send away. Explanation of arguments and prelude options can be found here.
- *Examples*. Many short examples to show its use of sxml.

2. The language

The language exists of two parts, a prelude and the document. The prelude is used to set options to introduce external modules or to define ways to output the resulting xml. The prelude is also optional.

The prelude starts with 3 dashes and ends with it. The prelude itself consists of a series of key-value pairs to control the way output is to be delivered. The prelude part is also optional. In that case defaults will be applied.

```
---
... prelude area ...
---
... document area ...
```

The document follows the prelude and is the thing you want to publish. It has a similar look as any xml typed language but is easier to read.

Below you see a simple example where the prelude is not used. When the prelude is missing the default action is to write the result into a second file. The name of the file is kept the same but with an extension of '.xml'.

```
$html [                                                    1
  $head [
    $title [Hello World]                                   2
  ]
  $body [
    $h1 [My first header]
    $p [
      And another greeting
    ]
  ]
]                                                          3
```

1. An element which becomes <html> with a body starting with '[' enclosing other elements and text.
2. The title element only contains text and no other elements.
3. The html element ends here with the ']' generating </html>.

The result after running through sxml2xml is shown below. Importend to note that this text is nicely formatted for this example and the output would be more like a one-liner.

```
<html>                                                     1
  <head>
    <title>Hello World</title>                             2
  </head>
  <body>
    <h1>My first header</h1>
    <p>And another greeting</p>
  </body>
</html>                                                    3
```

1. Result of '$html [' starts with <html>.
2. The title element is generated as <title>... text ... </title>.
3. The html element ends here with the generated </html>.

2.1. Document

The document is where the text comes in the same way as in a normal XML document. The elements most of us are aware of all start with a dollar '$'. Above in an example we have seen the expression *$title [Hello World]*. The body of the element is enclosed between square brackets '[' and ']'. Within these brackets you can write new elements intermittent with normal text. The methods in the library will not check if a particular element may be used on such place, you will need to look into the specifics of the XML language at hand.. So the the syntax of writing an element structure is quite simple.

However, sxml is capable of doing more than just entering XML elements in an alternative way. E.g. an element might need attributes. Below you see in a not too strict *BNF* variant a part of the syntax;

```
<document> ::= <prefix> <element> ( <attribute> '=' <attr-value> )*
               '[' <body-start-control> <body> <body-end-control>
```

```
                ']';

<prefix> ::= '$.' | '$!' | '$*<' | '$*>' | '$*' | '$';

<element> ::= <identifier>;
<attribute> ::= <identifier>;
<attr-value> ::= '"' <ws-text> '"' | "'" <ws-text> "'"
                  | <non-ws-text>;

<identifier> ::=  <letter> (<letter>|<number>)*
                   ( '-' (<letter> (<letter>|<number>)+) )*;

<body-start-control> ::= '!=' | '!' | '=';
<body-end-control> ::= '!';
```

White space is forbidden between the <prefix> and <element> as well as between the <attribute>, '=' and <attr-value>. Furthermore between the brackets and <body-start-control> and <body-end-control> are also no spaces. In the BNF above is not appearant that the '[!' combination must be used with the '!] combination.

2.1.1. Element types

2.1.2. Escaping characters

2.1.3. Unicode characters

2.1.4. Core elements

2.2. Prelude

The prelude is used to e.g. control to output of the result. Other usages are referencing libraries to be used in the document.

The prelude consists of a series of key-value pairs. The keys are defined as a series of catagories and subcatagories. The value can be anything. The key value pair is separated by a colon ':' and the pair ends in a ';'. The prelude part is also optional. In that case defaults will be applied.

Below you see an example of two options to let the xml prelude be written as well as a doctype at the start of the result.

```
---
option/xml-prelude/show:      1;
option/doctype/show:          1;
---
```

2.2.1. Options used by the library Semi-xml

module

option/doctype/definition

```
option/doctype/definition: [
  <!ENTITY company "Acme Mc Carpenter, Inc">
```

```
  <!ENTITY program "sxml2xml">
  <!ENTITY library "Semi-xml">
  <!ENTITY nbsp " ">
]
```

option/doctype/show

option/http-header

option/xml-prelude/encoding

option/xml-prelude/show

option/xml-prelude/version

output/fileext

output/filename

output/filepath

output/program

```
output/program/pdf:
  | xsltproc --encoding utf-8 --xinclude stylesheet.xsl -
  | xep -fo - -pdf sxml2xml.pdf
  ;
```

3. The programmers view

3.1. Classes data and methods

3.2. Substitution data

3.3. Methods

4. Using program sxml2xml

4.1. Program arguments

```
sxml2xml [--run=<run-selector>] <sxml-file>
```

4.2. Prelude options

4.2.1. dependencies/files

The value of this option is a list of paths to semi-xml documents which must be processed after the current one.

4.3. Unix she-bang usage

5. Syntax

```
<sxml-syntax> ::= <prelude-section>? <document>;

<prelude-section> ::= '---' <new-line> <key-value-pairs>* '---' <new-line>;
<key-value-pairs> ::= <key-name> ':' <value> ';';
<key-name> ::= <key-part> ('/' <key-part>)*;
<key-part> ::= <letter> (<letter>|<number>)*;

<document> ::= <prefix> <element> ( <attribute> '=' <attr-value> )*
               '[' <body-start-control> <body> <body-end-control>
               ']';

<prefix> ::= '$.' | '$!' | '$*<' | '$*>' | '$*' | '$';

<element> ::= <identifier>;
<attribute> ::= <identifier>;
<attr-value> ::= '"' <ws-text> '"' | "'" <ws-text> "'"
                 | <non-ws-text>;

<identifier> ::=  <letter> (<letter>|<number>)*
                  ( '-' (<letter> (<letter>|<number>)+) )*;

<body-start-control> ::= '!=' | '!' | '=';
<body-end-control> ::= '!';
```

6. Examples

Index