SWI-Prolog SGML/XML parser

Jan Wielemaker
SWI,
University of Amsterdam
The Netherlands
E-mail: jan@swi.psy.uva.nl

1 Introduction

Markup languages have recently regained popularity for two reasons. One is document exchange, which is largely based on HTML, an instance of SGML and the other is for data-exchange between programs, which is often based on XML, which can be considered simplified and rationalised version of SGML.

James Clark's SP parser is a flexible SGML and XML parser. Unfortunately it has some drawbacks. It is very big, not very fast, cannot work under event-driven input and is generally

```
          [],
          [ element(head,
                    [],
                    [ element(title,
                              [],
                              [ 'Demo'
                              ])
                    ]),
            element(body,
                    [],
                    [ '\n',
                      element(h1,
                              [ align = center
                              ],
                              [ 'This is a demo'
                              ]),
                      '\n\n',
                      element(p,
                              [],
                              [ 'Paragraphs in HTML need not be closed.\n'
                              ]),
                      element(p,
                              [],
                              [ 'This is called `omitted-tag\' handling.'
                              ])
                    ])
          ])
].
```

```
load_html_file(File, Term) :-
        dtd(html, DTD),
        load_structure(File, Term,
                       [ dtd(DTD),
                         dialect(sgml)
                       ]).
```

3 Predicate Reference

3.1 Loading Structured Documents

SGML or XML files are loaded through the common predicate `load_structure/3`. This is a predicate with many options. For simplicity a number of commonly used shorthands are provided: `load_sgml_file/2`, `load_xml_file/2`, and `load_html_file/2`.

load_structure(+*File*, {*ListOfContent*, +*Options*)

Load the XML file *File* and return the resulting structure in *ListOfContent*. *Options* is a list of options controlling the conversion process.

A proper XML document contains only a single toplevel element whose name matches the document type. Nevertheless, a list is returned for consistency with the representation of element content. The *ListOfContent* consists of three types:

Atom

Atoms are `CDATA`. Note is possible SWI-Prolog, as is no length-limit on atoms and atom garbage collection is provided.

element(*Name*, *ListAttributes*, *ListOfContent*)

Name

space(*sgml*)

In SGML, newlines at the start and end of an element are removed.[1] This is the default

White-space handling

White space mode is set to `preserve`. In addition to setting white-space handling at the toplevel the XML reserved attribute `<xml:space>`

3.4 DTD-Handling

element(*Name, Omit, Content*

notations(*ListOfNotations*)

Returns a list holding the names of all `NOTATION` declarations.

notation(*Name, File*)

Yields the declared file for from a `NOTATION` declaration.

3.5 Extracting a DTD

Some documents have no DTD. One of the neat facilities of this library is that it builds a DTD while parsing a document with an *implicit* DTD. The resulting DTD contains all elements encountered in the document. For each element the content model is a disjunction of elements and possibly `#PCDATA` that can be repeated. Thus, if in element <x> whe found element <y> and CDATA, the model is:

```
<!ELEMENT x - - (y|#PCDATA)*>
```

file(*File*)

Sets the file for reporting errors and warnings. Sets the line to 1.

line(*Line*)

Sets the current line. Useful if the stream is not at the start of the (file) object for goal(*+Goal*)

Goal is a callable term. The predicate `sgml_parse/2` opens an output stream to the cdata

CDATA has been parsed. The named handler is called with two arguments: *Handler*(+*CDATA*, +*Parser*), where CDATA is an atom representing the data.

entity

An entity that cannot be represented as CDATA has been parsed. The named handler is called with two arguments: *Handler*(+*NameOrCode*, +*Parser*).

pi

A processing instruction has been parsed. The named handler is called with two arguments: *Handler*(+*Text*, +*Parser*), where *Text* is the text of the processing instruction.

xmlns

Handler(

```
on_end('RDF', _) :-
```