The PiLLoW Web Programming Library

REFERENCE MANUAL
Generated/Printed on: 5 January 2001

Daniel Cabeza
Manuel Hermenegildo
clip@clip.dia.fi.upm.es
http://www.clip.dia.fi.upm.es/
The CLIP Group
School of Computer Science
Technical University of Madrid

Table of Contents

Summary **1**

1 Introduction **3**
- 1.1 Installing PiLLoW 3
- 1.2 Usage and interface (`pillow`) 3

2 HTML/XML/CGI programming **5**
- 2.1 Usage and interface (`html`) 5
- 2.2 Documentation on exports (`html`) 5
  - output_html/1 (pred) 5
  - html2terms/2 (pred) 5
  - xml2terms/2 (pred) 6
  - html_template/3 (pred) 6
  - html_report_error/1 (pred) 7
  - get_form_input/1 (pred) 7
  - get_form_value/3 (pred) 8
  - form_empty_value/1 (pred) 8
  - form_default/3 (pred) 8
  - set_cookie/2 (pred) 9
  - get_cookies/1 (pred) 9
  - url_query/2 (pred) 9
  - my_url/1 (pred) 9
  - url_info/2 (pred) 9
  - url_info_relative/3 (pred) 10
  - form_request_method/1 (pred) 11
  - icon_address/2 (pred) 11
  - html_protect/1 (pred) 11
  - http_lines/3 (pred) 11
- 2.3 Documentation on multifiles (`html`) 12
  - html_expansion/2 (pred) 12
- 2.4 Other information (`html`) 12

3 HTTP conectivity **13**
- 3.1 Usage and interface (`http`) 13
- 3.2 Documentation on exports (`http`) 13
  - fetch_url/3 (pred) 13

4 PiLLoW types **15**
- 4.1 Usage and interface (`pillow_types`) 15
- 4.2 Documentation on exports (`pillow_types`) 15
  - canonic_html_term/1 (regtype) 15
  - canonic_xml_term/1 (regtype) 16
  - html_term/1 (regtype) 17
  - form_dict/1 (regtype) 19
  - form_assignment/1 (regtype) 19
  - form_value/1 (regtype) 19
  - value_dict/1 (regtype) 20
  - url_term/1 (regtype) 20

http_request_param/1 (regtype) 20
http_response_param/1 (regtype) 20
http_date/1 (regtype) 21
weekday/1 (regtype) 21
month/1 (regtype) 21
hms_time/1 (regtype) 22

References 23

Predicate/Method Definition Index 25

Regular Type Definition Index 27

Global Index 29

Summary

The PiLLoW library ("Programming in Logic Languages on the Web") is a public domain Internet/WWW programming library for Logic Programming Systems which simplifies the process of writing applications for such environment. The library provides facilities for generating HTML or XML structured documents by handling them as Herbrand terms, producing HTML forms, writing form handlers, processing HTML templates, accessing and parsing WWW documents (either HTML or XML), accessing code posted at HTTP addresses, etc.

The PiLLoW library has been developed in the context of the &-Prolog and Ciao systems, but it has been adapted to a number of popular Logic Programming systems. Note, however, that this manual is written for the Ciao system, so if you use it in other systems, minor details in the documentation may not completely agree. For example, whereas in Ciao the library is a package which loads modules `html` and `http`, in the other systems it is just one module `pillow`. Thus, do not consider the *Library usage* information in other systems.

PiLLoW is distributed under the GNU Library General Public License.

1 Introduction

This package implements the PiLLoW library [CHV96]. The following three chapters document, respectively, the predicates for HTML/XML/CGI programming, the predicate for HTTP conectivity, and the types used in the definition of the predicates (key for fully understanding the other predicates). You can find a paper and some additional information in the `library/pillow/doc` directory of the distribution, and in the WWW at `http://clip.dia.fi.upm.es/Software/pillow/pillow.html`. There is also a *PiLLoW on-line tutorial* (slides) at `http://clip.dia.fi.upm.es/logalg/slides/C_pillow/C_pillow.html` which illustrates the basic features and provides a number of examples of PiLLoW use.

1.1 Installing PiLLoW

To correctly install PiLLoW, first, make sure you downloaded the right version of PiLLoW (there are different versions for different LP/CLP systems; the version that comes with Ciao is of course the right one for Ciao). Then, please follow these steps:

1. Copy the files in the `images` directory to a WWW accessible directory in your server.
2. Edit the file `icon_address.pl` and change the fact to point to the URL to be used to access the images above.
3. In the Ciao system the files are in the correct place, in other systems copy the files `pillow.pl` and `icon_address.pl` to a suitable directory so that your Prolog system will find them.

1.2 Usage and interface (`pillow`)

- **Library usage:**
  `:- use_package(pillow).`
  or
  `:- module(...,...,[pillow]).`
- **New operators defined:**
  `$/2` [150,xfx], `$/1` [150,fx].

2 HTML/XML/CGI programming

Author(s): Daniel Cabeza, Manuel Hermenegildo, Sacha Varma.

Version: 1.7#40 (2001/1/5, 19:7:40 CET)

Version of last change: 1.5#114 (2000/4/11, 20:23:43 CEST)

This module implements the predicates of the PiLLoW package related to HTML/ XML generation and parsing, CGI and form handlers programming, and in general all the predicates which do not imply the use of the HTTP protocol.

2.1 Usage and interface (`html`)

- **Library usage:**

  `:- use_module(library(html)).`
- **Exports:**
  - *Predicates:*

    `output_html/1`, `html2terms/2`, `xml2terms/2`, `html_template/3`, `html_report_error/1`, `get_form_input/1`, `get_form_value/3`, `form_empty_value/1`, `form_default/3`, `set_cookie/2`, `get_cookies/1`, `url_query/2`, `my_url/1`, `url_info/2`, `url_info_relative/3`, `form_request_method/1`, `icon_address/2`, `html_protect/1`, `http_lines/3`.
  - *Multifiles:*

    `html_expansion/2`.

2.2 Documentation on exports (`html`)

output_html/1: PREDICATE

`output_html(HTMLTerm)`

Outputs `HTMLTerm`, interpreted as an `html_term/1`, to current output stream.

Usage:

- *The following properties should hold at call time:*

  `HTMLTerm` is a term representing HTML code. (`html_term/1`)

html2terms/2: PREDICATE

`html2terms(String,Terms)`

`String` is a character list containing HTML code and `Terms` is its prolog structured representation.

Usage 1:

- *Description:* Translates an HTML-term into the HTML code it represents.
- *The following properties should hold at call time:*

  `String` is a free variable. (`var/1`)

  `Terms` is a term representing HTML code. (`html_term/1`)
- *The following properties hold upon exit:*

  `String` is a string (a list of character codes). (`string/1`)

Usage 2:

- *Description:* Translates HTML code into a structured HTML-term.
- *Calls should, and exit will be compatible with:*
  `Terms` is a term representing HTML code in canonical form. (`canonic_html_term/1`)
- *The following properties should hold at call time:*
  `String` is a string (a list of character codes). (`string/1`)
- *The following properties hold upon exit:*
  `Terms` is a term representing HTML code in canonical form. (`canonic_html_term/1`)

xml2terms/2: PREDICATE

`xml2terms(String,Terms)`

`String` is a character list containing XML code and `Terms` is its prolog structured representation.

Usage 1:

- *Description:* Translates a XML-term into the XML code it represents.
- *The following properties should hold at call time:*
  `String` is a free variable. (`var/1`)
  `Terms` is a term representing HTML code. (`html_term/1`)
- *The following properties hold upon exit:*
  `String` is a string (a list of character codes). (`string/1`)

Usage 2:

- *Description:* Translates XML code into a structured XML-term.
- *Calls should, and exit will be compatible with:*
  `Terms` is a term representing XML code in canonical form. (`canonic_xml_term/1`)
- *The following properties should hold at call time:*
  `String` is a string (a list of character codes). (`string/1`)
- *The following properties hold upon exit:*
  `Terms` is a term representing XML code in canonical form. (`canonic_xml_term/1`)

html_template/3: PREDICATE

`html_template(Chars,Terms,Dict)`

Interprets `Chars` as an HTML template returning in `Terms` the corresponding structured HTML-term, which includes variables, and unifying `Dict` with a dictionary of those variables (an incomplete list of *name*`=`*Var* pairs). An HTML template is standard HTML code, but in which "slots" can be defined and given an identifier. These slots represent parts of the HTML code in which other HTML code can be inserted, and are represented in the HTML-term as free variables. There are two kinds of variables in templates:

- Variables representing page contents. A variable with name *name* is defined with the special tag `<V>`*name*`</V>`.
- Variables representing tag attributes. They occur as an attribute or an attribute value starting with `_`, followed by its name, which must be formed by alphabetic characters.

As an example, suposse the following HTML template:

```
<html>
<body bgcolor=_bgcolor>
<v>content</v>
</body>
</html>
```

The following query in the Ciao toplevel shows how the template is parsed, and the dictionary returned:

```
?- file_to_string('template.html',_S), html_template(_S,Terms,Dict).

Dict = [bgcolor=_A,content=_B|_],
Terms = [env(html,[],["
",env(body,[bgcolor=_A],["
",_B,"
"]),"
"]),"
"] ?

yes
```

If a dictionary with values is supplied at call time, then variables are unified accordingly inside the template:

```
?- file_to_string('template.html',_S),
   html_template(_S,Terms,[content=b("hello world!"),bgcolor="white"]).

Terms = [env(html,[],["
",env(body,[bgcolor="white"],["
",b("hello world!"),"
"]),"
"]),"
"] ?

yes
```

Usage:

- *Calls should, and exit will be compatible with:*
  `Terms` is a term representing HTML code in canonical form. (`canonic_html_term/1`)
  `Dict` is a list. (`list/1`)
- *The following properties should hold at call time:*
  `Chars` is a string (a list of character codes). (`string/1`)
- *The following properties hold upon exit:*
  `Terms` is a term representing HTML code in canonical form. (`canonic_html_term/1`)
  `Dict` is a list. (`list/1`)

html_report_error/1: PREDICATE

Usage: `html_report_error(Error)`

- *Description:* Outputs error `Error` as a standard HTML page.

get_form_input/1: PREDICATE

`get_form_input(Dict)`

Translates input from the form (with either the POST or GET methods, and even with CONTENT_TYPE multipart/form-data) to a dictionary `Dict` of *attribute=value* pairs. It translates empty values (which indicate only the presence of an attribute) to the atom `'$empty'`, values with more than one line (from text areas or files) to a list of lines as strings, the rest to atoms or numbers (using `name/2`).

Usage:

- *The following properties should hold at call time:*

  `Dict` is a free variable. (`var/1`)
- *The following properties hold upon exit:*

  `Dict` is a dictionary of values of the attributes of a form. It is a list of `form_assignment` (`form_dict/1`)

get_form_value/3: PREDICATE

`get_form_value(Dict,Var,Val)`

Unifies `Val` with the value for attribute `Var` in dictionary `Dict`. Does not fail: value is `''` if not found (this simplifies the programming of form handlers when they can be accessed directly).

Usage:

- *Calls should, and exit will be compatible with:*

  `Val` is a value of an attribute of a form. (`form_value/1`)
- *The following properties should hold at call time:*

  `Dict` is a dictionary of values of the attributes of a form. It is a list of `form_assignment` (`form_dict/1`)

  `Var` is an atom. (`atm/1`)
- *The following properties hold upon exit:*

  `Val` is a value of an attribute of a form. (`form_value/1`)

form_empty_value/1: PREDICATE

Usage: `form_empty_value(Term)`

- *Description:* Checks that `Term`, a value comming from a text area is empty (can have spaces, newlines and linefeeds).

form_default/3: PREDICATE

Usage: `form_default(Val,Default,NewVal)`

- *Description:* Useful when a form is only partially filled, or when the executable can be invoked either by a link or by a form, to set form defaults. If the value of `Val` is empty then `NewVal=Default`, else `NewVal=Val`.
- *The following properties should hold at call time:*

  `Val` is currently a term which is not a free variable. (`nonvar/1`)

  `Default` is currently a term which is not a free variable. (`nonvar/1`)

  `NewVal` is a free variable. (`var/1`)

set_cookie/2: PREDICATE

`set_cookie(Name,Value)`

Sets a cookie of name `Name` and value `Value`. Must be invoked before outputting any data, including the `cgi_reply` html-term.

Usage:

- *The following properties should hold at call time:*

  `Name` is an atom. (`atm/1`)

  `Value` is an atomic term (an atom or a number). (`constant/1`)

get_cookies/1: PREDICATE

`get_cookies(Cookies)`

Unifies `Cookies` with a dictionary of *attribute=value* pairs of the active cookies for this URL.

Usage:

- *The following properties should hold at call time:*

  `Cookies` is a free variable. (`var/1`)
- *The following properties hold upon exit:*

  `Cookies` is a dictionary of values. It is a list of pairs *atom=constant*. (`value_dict/1`)

url_query/2: PREDICATE

`url_query(Dict,URLArgs)`

Translates a dictionary `Dict` of parameter values into a string `URLArgs` for appending to a URL pointing to a form handler.

Usage:

- *The following properties should hold at call time:*

  `Dict` is a dictionary of values. It is a list of pairs *atom=constant*. (`value_dict/1`)

  `URLArgs` is a free variable. (`var/1`)
- *The following properties hold upon exit:*

  `URLArgs` is a string (a list of character codes). (`string/1`)

my_url/1: PREDICATE

`my_url(URL)`

Unifies `URL` with the Uniform Resource Locator (WWW address) of this cgi executable.

Usage:

- *Calls should, and exit will be compatible with:*

  `URL` is a string (a list of character codes). (`string/1`)
- *The following properties hold upon exit:*

  `URL` is a string (a list of character codes). (`string/1`)

url_info/2: PREDICATE

`url_info(URL,URLTerm)`

Translates a URL `URL` to a Prolog structure `URLTerm` which details its various components, and vice-versa. For now non-HTTP URLs make the predicate fail.

Usage 1:

- *Calls should, and exit will be compatible with:*
  `URLTerm` specifies a URL. (`url_term/1`)
- *The following properties should hold at call time:*
  `URL` is an atom. (`atm/1`)
- *The following properties hold upon exit:*
  `URLTerm` specifies a URL. (`url_term/1`)

Usage 2:

- *Calls should, and exit will be compatible with:*
  `URLTerm` specifies a URL. (`url_term/1`)
- *The following properties should hold at call time:*
  `URL` is a string (a list of character codes). (`string/1`)
- *The following properties hold upon exit:*
  `URLTerm` specifies a URL. (`url_term/1`)

Usage 3:

- *The following properties should hold at call time:*
  `URL` is a free variable. (`var/1`)
  `URLTerm` specifies a URL. (`url_term/1`)
- *The following properties hold upon exit:*
  `URL` is a string (a list of character codes). (`string/1`)

url_info_relative/3: PREDICATE

`url_info_relative(URL,BaseURLTerm,URLTerm)`

Translates a relative URL `URL` which appears in the HTML page refered to by `BaseURLTerm` into `URLTerm`, a Prolog structure containing its absolute parameters. Absolute URLs are translated as with `url_info/2`. E.g.

```
url_info_relative("dadu.html",
                  http('www.foo.com',80,"/bar/scoob.html"), Info)
```

gives `Info = http('www.foo.com',80,"/bar/dadu.html")`.

Usage 1:

- *Calls should, and exit will be compatible with:*
  `URLTerm` specifies a URL. (`url_term/1`)
- *The following properties should hold at call time:*
  `URL` is an atom. (`atm/1`)
  `BaseURLTerm` specifies a URL. (`url_term/1`)
- *The following properties hold upon exit:*
  `URLTerm` specifies a URL. (`url_term/1`)

Usage 2:

– *Calls should, and exit will be compatible with:*
`URLTerm` specifies a URL. (`url_term/1`)
– *The following properties should hold at call time:*
`URL` is a string (a list of character codes). (`string/1`)
`BaseURLTerm` specifies a URL. (`url_term/1`)
– *The following properties hold upon exit:*
`URLTerm` specifies a URL. (`url_term/1`)

form_request_method/1: PREDICATE

Usage: `form_request_method(Method)`
– *Description:* Unifies `Method` with the method of invocation of the form handler (`GET` or `POST`).
– *The following properties hold upon exit:*
`Method` is an atom. (`atm/1`)

icon_address/2: PREDICATE

`icon_address(Img,IAddress)`
The PiLLoW image `Img` has URL `IAddress`.
Usage:
– *Calls should, and exit will be compatible with:*
`Img` is an atom. (`atm/1`)
`IAddress` is an atom. (`atm/1`)
– *The following properties hold upon exit:*
`Img` is an atom. (`atm/1`)
`IAddress` is an atom. (`atm/1`)

html_protect/1: PREDICATE

`html_protect(Goal)`
Calls `Goal`. If an error occurs during its execution, or it fails, an HTML page is output informing about the incident. Normaly the whole execution of a CGI is protected thus.
Meta-predicate with arguments: `html_protect(goal)`.
Usage:
– *Calls should, and exit will be compatible with:*
`Goal` is a term which represents a goal, i.e., an atom or a structure. (`callable/1`)

http_lines/3: PREDICATE

Usage: `http_lines(Lines,String,Tail)`
– *Description:* `Lines` is a list of the lines with occur in `String` until `Tail`. The lines may end UNIX-style or DOS-style in `String`, in `Lines` they have not end of line characters. Suitable to be used in DCGs.
– *Calls should, and exit will be compatible with:*
`Lines` is a list of `strings`. (`list/2`)
`String` is a string (a list of character codes). (`string/1`)
`Tail` is a string (a list of character codes). (`string/1`)

2.3 Documentation on multifiles (`html`)

html_expansion/2: PREDICATE

The predicate is *multifile*.

Usage: `html_expansion(Term,Expansion)`

- *Description:* Hook predicate to define macros. Expand occurrences of `Term` into `Expansion`, in `output_html/1`. Take care to not transform something into itself!

2.4 Other information (`html`)

The code uses input from from L. Naish's forms and F. Bueno's previous Chat interface. Other people who have contributed is (please inform us if we leave out anybody): Markus Fromherz, Samir Genaim.

3 HTTP conectivity

Author(s): Daniel Cabeza.

Version: 1.7#40 (2001/1/5, 19:7:40 CET)

Version of last change: 1.3#114 (1999/11/24, 0:57:16 MET)

This module implements the HTTP protocol, which allows retrieving data from HTTP servers.

3.1 Usage and interface (`http`)

- **Library usage:**
  `:- use_module(library(http)).`
- **Exports:**
  - *Predicates:*
    `fetch_url/3`.

3.2 Documentation on exports (`http`)

fetch_url/3: PREDICATE

`fetch_url(URL,Request,Response)`

Fetches the document pointed to by `URL` from Internet, using request parameters `Request`, and unifies `Response` with the parameters of the response. Fails on timeout. Note that redirections are not handled automatically, that is, if `Response` contains terms of the form `status(redirection,301,_)` and `location(NewURL)`, the program should in most cases access location `NewURL`.

Usage: `fetch_url(URL,Request,Response)`

- *The following properties should hold at call time:*
  `URL` specifies a URL. (`url_term/1`)
  `Request` is a list of `http_request_params`. (`list/2`)
- *The following properties hold upon exit:*
  `Response` is a list of `http_response_params`. (`list/2`)

4 PiLLoW types

Author(s): Daniel Cabeza.

Here are defined the regular types used in the documentation of the predicates of the PiLLoW package.

4.1 Usage and interface (pillow_types)

- **Library usage:**
  `:- use_module(library(pillow_types)).`
- **Exports:**
  - *Regular Types:*
    `canonic_html_term/1`, `canonic_xml_term/1`, `html_term/1`, `form_dict/1`, `form_assignment/1`, `form_value/1`, `value_dict/1`, `url_term/1`, `http_request_param/1`, `http_response_param/1`, `http_date/1`, `weekday/1`, `month/1`, `hms_time/1`.

4.2 Documentation on exports (pillow_types)

canonic_html_term/1: REGTYPE

A term representing HTML code in canonical, structured way. It is a list of terms defined by the following predicate:

```
canonic_html_item(comment(S)) :-
        string(S).
canonic_html_item(declare(S)) :-
        string(S).
canonic_html_item(env(Tag,Atts,Terms)) :-
        atm(Tag),
        list(Atts,tag_attrib),
        canonic_html_term(Terms).
canonic_html_item($(Tag,Atts)) :-
        atm(Tag),
        list(Atts,tag_attrib).
canonic_html_item(S) :-
        string(S).
tag_attrib(Att) :-
        atm(Att).
tag_attrib(Att=Val) :-
        atm(Att),
        string(Val).
```

Each structure represents one HTML construction:

env(*tag***,***attribs***,***terms***)**
: An HTML environment, with name *tag*, list of attributes *attribs* and contents *terms*.

$(*tag***,***attribs***)**
: An HTML element of name *tag* and list of attributes *attribs*. `($)/2` is defined by the pillow package as an infix, binary operator.

comment(*string*)
: An HTML comment (translates to/from `<!--`*string*`-->`).

declare(*string*)
: An HTML declaration, they are used only in the header (translates to/from `<!`*string*`>`).

string Normal text is represented as a list of character codes.

For example, the term

```
env(a,[href="www.therainforestsite.com"],
      ["Visit ",img$[src="TRFS.gif"]])
```

is output to (or parsed from):

```
<a href="www.therainforestsite.com">Visit <img src="TRFS.gif"></a>
```

Usage: `canonic_html_term(HTMLTerm)`

– *Description:* `HTMLTerm` is a term representing HTML code in canonical form.

canonic_xml_term/1: REGTYPE

A term representing XML code in canonical, structured way. It is a list of terms defined by the following predicate (see `tag_attrib/1` definition in `canonic_html_term/1`):

```
canonic_xml_item(Term) :-
        canonic_html_item(Term).
canonic_xml_item(xmldecl(Atts)) :-
        list(Atts,tag_attrib).
canonic_xml_item(env(Tag,Atts,Terms)) :-
        atm(Tag),
        list(Atts,tag_attrib),
        canonic_xml_term(Terms).
canonic_xml_item(elem(Tag,Atts)) :-
        atm(Tag),
        list(Atts,tag_attrib).
```

In addition to the structures defined by `canonic_html_term/1` (the `($)/2` structure appears only in malformed XML code), the following structures can be used:

elem(*tag*,*atts*)
: Specifies an XML empty element of name *tag* and list of attributes *atts*. For example, the term

```
elem(arc,[weigh="3",begin="n1",end="n2"])
```

is output to (or parsed from):

```
<arc weigh="3" begin="n1" end="n2"/>
```

xmldecl(*atts*)
: Specifies an XML declaration with attributes *atts* (translates to/from `<?xml` *atts*`?>`)

Usage: `canonic_xml_term(XMLTerm)`

– *Description:* `XMLTerm` is a term representing XML code in canonical form.

html_term/1: REGTYPE

A term which represents HTML or XML code in a structured way. In addition to the structures defined by `canonic_html_term/1` or `canonic_xml_term/1`, the following structures can be used:

begin(*tag*,*atts***)**
: It translates to the start of an HTML environment of name *tag* and attributes *atts*. There exists also a **begin(***tag***)** structure. Useful, in conjunction with the next structure, when including in a document output generated by an existing piece of code (e.g. *tag* = `pre`). Its use is otherwise discouraged.

end(*tag***)** Translates to the end of an HTML environment of name *tag*.

start Used at the beginning of a document (translates to `<html>`).

end Used at the end of a document (translates to `</html>`).

-- Produces a horizontal rule (translates to `<hr>`).

\\ Produces a line break (translates to `<br>`).

$ Produces a paragraph break (translates to `<p>`).

image(*address***)**
: Used to include an image of address (URL) *address* (equivalent to `img$[src=`*address*`]`).

image(*address*,*atts***)**
: As above with the list of attributes *atts*.

ref(*address*,*text***)**
: Produces a hypertext link, *address* is the URL of the referenced resource, *text* is the text of the reference (equivalent to `a([href=`*address*`],`*text*`)`).

label(*name*,*text***)**
: Labels *text* as a target destination with label *name* (equivalent to `a([name=`*name*`],`*text*`)`).

heading(*n*,*text***)**
: Produces a heading of level *n* (between 1 and 6), *text* is the text to be used as heading. Useful when one wants a heading level relative to another heading (equivalent to `h`*n*`(`*text*`)`).

itemize(*items***)**
: Produces a list of bulleted items, *items* is a list of corresponding HTML terms (translates to a `<ul>` environment).

enumerate(*items***)**
: Produces a list of numbered items, *items* is a list of corresponding HTML terms (translates to a `<ol>` environment).

description(*defs***)**
: Produces a list of defined items, *defs* is a list whose elements are definitions, each of them being a Prolog sequence (composed by `','/2` operators). The last element of the sequence is the definition, the other (if any) are the defined terms (translates to a `<dl>` environment).

nice_itemize(*img*,*items***)**
: Produces a list of bulleted items, using the image *img* as bullet. The predicate `icon_address/2` provides a colored bullet.

preformatted(_text_**)**
Used to include preformatted text, _text_ is a list of HTML terms, each element of the list being a line of the resulting document (translates to a `<pre>` environment).

verbatim(_text_**)**
Used to include text verbatim, special HTML characters (`<`,`>`,`&`,`"` and space) are translated into its quoted HTML equivalent.

prolog_term(_term_**)**
Includes any prolog term _term_, represented in functional notation. Variables are output as `_`.

nl Used to include a newline in the HTML source (just to improve human readability).

entity(_name_**)**
Includes the entity of name _name_ (ISO-8859-1 special character).

start_form(_addr_,_atts_**)**
Specifies the beginning of a form. _addr_ is the address (URL) of the program that will handle the form, and _atts_ other attributes of the form, as the method used to invoke it. If _atts_ is not present (there is only one argument) the method defaults to POST.

start_form Specifies the beginning of a form without assigning address to the handler, so that the form handler will be the cgi-bin executable producing the form.

end_form Specifies the end of a form.

checkbox(_name_,_state_**)**
Specifies an input of type `checkbox` with name _name_, _state_ is `on` if the checkbox is initially checked.

radio(_name_,_value_,_selected_**)**
Specifies an input of type `radio` with name _name_ (several radio buttons which are interlocked must share their name), _value_ is the the value returned by the button, if _selected_=_value_ the button is initially checked.

input(_type_,_atts_**)**
Specifies an input of type _type_ with a list of attributes _atts_. Possible values of _type_ are `text`, `hidden`, `submit`, `reset`, ldots

textinput(_name_,_atts_,_text_**)**
Specifies an input text area of name _name_. _text_ provides the default text to be shown in the area, _atts_ a list of attributes.

option(_name_,_val_,_options_**)**
Specifies a simple option selector of name _name_, _options_ is the list of available options and _val_ is the initial selected option (if _val_ is not in _options_ the first item is selected by default) (translates to a `<select>` environment).

menu(_name_,_atts_,_items_**)**
Specifies a menu of name _name_, list of attributes _atts_ and list of options _items_. The elements of the list _items_ are marked with the prefix operator `$` to indicate that they are selected (translates to a `<select>` environment).

form_reply

cgi_reply This two are equivalent, they do not generate HTML, rather, the CGI protocol requires this content descriptor to be used at the beginning by CGI executables (including form handlers) when replying (translates to `Content-type: text/html`).

pr Includes in the page a graphical logo with the message "Developed using the PiLLoW Web programming library", which points to the manual and library source.

name(*text*)
A term with functor *name*/1, different from the special functors defined herein, represents an HTML environment of name *name* and included text *text*. For example, the term

```
address('clip@clip.dia.fi.upm.es')
```

is translated into the HTML source

```
<address>clip@clip.dia.fi.upm.es</address>
```

name(*atts*,*text*)
A term with functor *name*/2, different from the special functors defined herein, represents an HTML environment of name *name*, attributes *atts* and included text *text*. For example, the term

```
a([href='http://www.clip.dia.fi.upm.es/'],"Clip home")
```

represents the HTML source

```
<a href="http://www.clip.dia.fi.upm.es/">Clip home</a>
```

Usage: `html_term(HTMLTerm)`

– *Description:* `HTMLTerm` is a term representing HTML code.

form_dict/1: REGTYPE

Usage: `form_dict(Dict)`

– *Description:* `Dict` is a dictionary of values of the attributes of a form. It is a list of `form_assignment`

form_assignment/1: REGTYPE

Usage: `form_assignment(Eq)`

– *Description:* `Eq` is an assignment of value of an attribute of a form. It is defined by:

```
form_assignment(A=V) :-
        atm(A),
        form_value(V).
form_value(A) :-
        atm(A).
form_value(N) :-
        num(N).
form_value(L) :-
        list(L,string).
```

form_value/1: REGTYPE

Usage: `form_value(V)`

– *Description:* `V` is a value of an attribute of a form.

value_dict/1: REGTYPE

Usage: `value_dict(Dict)`

– *Description:* `Dict` is a dictionary of values. It is a list of pairs *atom=constant*.

url_term/1: REGTYPE

A term specifying an Internet Uniform Resource Locator. Currently only HTTP URLs are supported. Example: `http('www.clip.dia.fi.upm.es',80,"/Software/Ciao/")`. Defined as

```
url_term(http(Host,Port,Document)) :-
        atm(Host),
        int(Port),
        string(Document).
```

Usage: `url_term(URL)`

– *Description:* `URL` specifies a URL.

http_request_param/1: REGTYPE

A parameter of an HTTP request:

- **head:** Specify that the document content is not wanted.
- **timeout(***T***):** *T* specifies the time in seconds to wait for the response. Default is 300 seconds.
- **if_modified_since(***Date***):** Get document only if newer than *Date*. *Date* has the format defined by `http_date/1`.
- **user_agent(***Agent***):** Provides a user-agent field, *Agent* is an atom. The string `"PiLLoW/1.1"` (or whatever version of PiLLoW is used) is appended.
- **authorization(***Scheme*,*Params***):** To provide credentials. See RFC 1945 for details.
- *option*(*Value*)**:** Any unary term, being *Value* an atom, can be used to provide another valid option (e.g. `from('user@machine')`).

Usage: `http_request_param(Request)`

– *Description:* `Request` is a parameter of an HTTP request.

http_response_param/1: REGTYPE

A parameter of an HTTP response:

- **content(***String***):** *String* is the document content (list of bytes). If the `head` parameter of the HTTP request is used, an empty list is get here.
- **status(***Type*,*Code*,*Reason***):** *Type* is an atom denoting the response type, *Code* is the status code (an integer), and *Reason* is a string holding the reason phrase.
- **message_date(***Date***):** *Date* is the date of the response, with format defined by `http_date/1`.

- **location(*Loc*):** This parameter appears when the document has moved, *Loc* is an atom holding the new location.
- **http_server(*Server*):** *Server* is the server responding, as a string.
- **authenticate(*Params*):** Returned if document is protected, *Params* is a list of chagenges. See RFC 1945 for details.
- **allow(*Methods*):** *Methods* are the methods allowed by the server, as a list of atoms.
- **content_encoding(*Encoding*):** *Encoding* is an atom defining the encoding.
- **content_length(*Length*):** *Length* is the length of the document (an integer).
- **content_type(*Type,Subtype,Params*):** Specifies the document content type, *Type* and *Subtype* are atoms, *Params* a list of parameters (e.g. `content_type(text,html,[])`).
- **expires(*Date*):** *Date* is the date after which the entity should be considered stale. Format defined by `http_date/1`.
- **last_modified(*Date*):** *Date* is the date at which the sender believes the resource was last modified. Format defined by `http_date/1`.
- **pragma(*String*):** Miscellaneous data.
- *header*(*String*)**:** Any other functor *header*/1 is an extension header.

Usage: `http_response_param(Response)`

– *Description:* `Response` is a parameter of an HTTP response.

http_date/1: REGTYPE

`http_date(Date)`

`Date` is a term defined as

```
http_date(date(WeekDay,Day,Month,Year,Time)) :-
        weekday(WeekDay),
        int(Day),
        month(Month),
        int(Year),
        hms_time(Time).
```

.

Usage: `http_date(Date)`

– *Description:* `Date` is a term denoting a date.

weekday/1: REGTYPE

A regular type, defined as follows:

```
weekday('Monday').
weekday('Tuesday').
weekday('Wednesday').
weekday('Thursday').
weekday('Friday').
weekday('Saturday').
weekday('Sunday').
```

month/1: REGTYPE

A regular type, defined as follows:

```
month('January').
month('February').
month('March').
month('April').
month('May').
month('June').
month('July').
month('August').
month('September').
month('October').
month('November').
month('December').
```

hms_time/1: REGTYPE

Usage: `hms_time(Time)`

- *Description:* `Time` is an atom of the form `hh:mm:ss`

References

[**CHV96**] D. Cabeza, M. Hermenegildo, and S. Varma.
The PiLLoW/CIAO Library for INTERNET/WWW Programming using Computational Logic Systems.
In *Proceedings of the 1st Workshop on Logic Programming Tools for INTERNET Applications*, pages 72–90, JICSLP'96, Bonn, September 1996.

Predicate/Method Definition Index

F

fetch_url/3 13
form_default/3 8
form_empty_value/1 8
form_request_method/1 11

G

get_cookies/1 9
get_form_input/1 7
get_form_value/3 8

H

html_expansion/2 12
html_protect/1 11
html_report_error/1 7
html_template/3 6
html2terms/2 5
http_lines/3 11

I

icon_address/2 11

M

my_url/1 9

O

output_html/1 5

S

set_cookie/2 9

U

url_info/2 9
url_info_relative/3 10
url_query/2 9

X

xml2terms/2 6

Regular Type Definition Index

C

canonic_html_term/1 15
canonic_xml_term/1 16

F

form_assignment/1 19
form_dict/1 19
form_value/1 19

H

hms_time/1 22
html_term/1 17
http_date/1 21
http_request_param/1 20
http_response_param/1 20

M

month/1 21

U

url_term/1 20

V

value_dict/1 20

W

weekday/1 21

Global Index

This is a global index containing pointers to places where concepts, predicates, modes, properties, types, applications, etc., are referred to in the text of the document. Note that due to limitations of the info format unfortunately only the first reference will appear in online versions of the document.

$

$/1 3
$/2 3

C

canonic_html_term/1 .... 15, 16, 17
canonic_xml_term/1 .... 15, 16, 17
CGI 3, 5

F

fetch_url/3 13
form_assignment/1 .... 15, 19
form_default/3 .... 5, 8
form_dict/1 .... 15, 19
form_empty_value/1 .... 5, 8
form_request_method/1 .... 5, 11
form_value/1 .... 15, 19

G

get_cookies/1 .... 5, 9
get_form_input/1 .... 5, 7
get_form_value/3 .... 5, 8
GNU Library General Public License 1

H

hms_time/1 .... 15, 22
HTML 3, 5
html_expansion/2 .... 5, 12
html_protect/1 .... 5, 11
html_report_error/1 .... 5, 7
html_template/3 .... 5, 6
html_term/1 .... 5, 15, 17
html2terms/2 5
HTTP 3, 13
http_date/1 .... 15, 20, 21
http_lines/3 .... 5, 11
http_request_param/1 .... 15, 20
http_response_param/1 .... 15, 20

I

icon_address.pl 3
icon_address/2 .... 5, 11, 17

L

library/pillow/doc 3

M

month/1 .... 15, 21
my_url/1 .... 5, 9

N

name/2 8

O

output_html/1 .... 5, 12

P

PiLLoW on-line tutorial 3
pillow.pl 3

S

set_cookie/2 .... 5, 9

T

tag_attrib/1 16

U

url_info/2 .... 5, 9, 10
url_info_relative/3 .... 5, 10
url_query/2 .... 5, 9
url_term/1 .... 15, 20

V

value_dict/1 .... 15, 20

W

weekday/1 .... 15, 21
WWW, interfacing with 3

X

XML 3, 5
xml2terms/2 .... 5, 6