

HTML: Parsing Library

Version 5.0

June 6, 2010

```
(require html)
```

The `html` library provides functions to read html documents and structures to represent them.

```
(read-xhtml port) → html?  
  port : input-port?  
(read-html port) → html?  
  port : input-port?
```

Reads (X)HTML from a port, producing an `html` instance.

```
(read-html-as-xml port) → (listof content/c)  
  port : input-port?
```

Reads HTML from a port, producing an X-expression compatible with the `xml` library (which defines `content/c`).

```
(read-html-comments) → boolean?  
(read-html-comments v) → void?  
  v : any/c
```

If `v` is not `#f`, then comments are read and returned. Defaults to `#f`.

```
(use-html-spec) → boolean?  
(use-html-spec v) → void?  
  v : any/c
```

If `v` is not `#f`, then the HTML must respect the HTML specification with regards to what elements are allowed to be the children of other elements. For example, the top-level `"<html>"`

element may only contain a "`<body>`" and "`<head>`" element. Defaults to `#f`.

1 Example

```
(module html-example racket

  ; Some of the symbols in html and xml conflict with
  ; each other and with racket/base language, so we prefix
  ; to avoid namespace conflict.
  (require (prefix-in h: html)
           (prefix-in x: xml))

  (define an-html
    (h:read-xhtml
     (open-input-string
      (string-append
       "<html><head><title>My title</title></head><body>"
       "<p>Hello world</p><p><b>Testing</b>!</p>"
       "</body></html>"))))

  ; extract-pcdata: html-content -> (listof string)
  ; Pulls out the pcdata strings from some-content.
  (define (extract-pcdata some-content)
    (cond [(x:pcdata? some-content)
           (list (x:pcdata-string some-content))]
          [(x:entity? some-content)
           (list)]
          [else
           (extract-pcdata-from-element some-content)]))

  ; extract-pcdata-from-element: html-element -> (listof string)
  ; Pulls out the pcdata strings from an-html-element.
  (define (extract-pcdata-from-element an-html-element)
    (match an-html-element
      [(struct h:html-full (attributes content))
       (apply append (map extract-pcdata content))]

      [(struct h:html-element (attributes))
       '()])))

  (printf "~s~n" (extract-pcdata an-html)))

> (require 'html-example)
("My title" "Hello world" "Testing" "!")
```

2 HTML Structures

`pcdata`, `entity`, and `attribute` are defined in the `xml` documentation.

A `html-content` is either

- `html-element`
- `pcdata`
- `entity`

```
(struct html-element (attributes)  
      #:extra-constructor-name make-html-element)  
  attributes : (listof attribute)
```

Any of the structures below inherits from `html-element`.

```
(struct html-full struct:html-element (content)  
      #:extra-constructor-name make-html-full)  
  content : (listof html-content)
```

Any html tag that may include content also inherits from `html-full` without adding any additional fields.

```
(struct html html-full ()  
      #:extra-constructor-name make-html)
```

A `html` is (`make-html` (`listof attribute`) (`listof Contents-of-html`))

A `Contents-of-html` is either

- `body`
- `head`

```
(struct div html-full ()  
      #:extra-constructor-name make-div)
```

A `div` is (`make-div` (`listof attribute`) (`listof G2`))

```
(struct center html-full ()  
      #:extra-constructor-name make-center)
```

A `center` is `(make-center (listof attribute) (listof G2))`

```
(struct blockquote html-full ()  
  #:extra-constructor-name make-blockquote)
```

A `blockquote` is `(make-blockquote (listof attribute) G2)`

```
(struct ins html-full ()  
  #:extra-constructor-name make-ins)
```

An `Ins` is `(make-ins (listof attribute) (listof G2))`

```
(struct del html-full ()  
  #:extra-constructor-name make-del)
```

A `del` is `(make-del (listof attribute) (listof G2))`

```
(struct dd html-full ()  
  #:extra-constructor-name make-dd)
```

A `dd` is `(make-dd (listof attribute) (listof G2))`

```
(struct li html-full ()  
  #:extra-constructor-name make-li)
```

A `li` is `(make-li (listof attribute) (listof G2))`

```
(struct th html-full ()  
  #:extra-constructor-name make-th)
```

A `th` is `(make-th (listof attribute) (listof G2))`

```
(struct td html-full ()  
  #:extra-constructor-name make-td)
```

A `td` is `(make-td (listof attribute) (listof G2))`

```
(struct iframe html-full ()  
  #:extra-constructor-name make-iframe)
```

An `iframe` is `(make-iframe (listof attribute) (listof G2))`

```
(struct noframes html-full ()
  #:extra-constructor-name make-noframes)

A noframes is (make-noframes (listof attribute) (listof G2))
```

```
(struct noscript html-full ()
  #:extra-constructor-name make-noscript)

A noscript is (make-noscript (listof attribute) (listof G2))
```

```
(struct style html-full ()
  #:extra-constructor-name make-style)

A style is (make-style (listof attribute) (listof pcddata))
```

```
(struct script html-full ()
  #:extra-constructor-name make-script)

A script is (make-script (listof attribute) (listof pcddata))
```

```
(struct basefont html-element ()
  #:extra-constructor-name make-basefont)

A basefont is (make-basefont (listof attribute))
```

```
(struct br html-element ()
  #:extra-constructor-name make-br)

A br is (make-br (listof attribute))
```

```
(struct area html-element ()
  #:extra-constructor-name make-area)

An area is (make-area (listof attribute))
```

```
(struct alink html-element ()
  #:extra-constructor-name make-alink)

A alink is (make-alink (listof attribute))
```

```
(struct img html-element ())  
  #:extra-constructor-name make-img)
```

An `img` is (`make-img` (`listof attribute`))

```
(struct param html-element ())  
  #:extra-constructor-name make-param)
```

A `param` is (`make-param` (`listof attribute`))

```
(struct hr html-element ())  
  #:extra-constructor-name make-hr)
```

A `hr` is (`make-hr` (`listof attribute`))

```
(struct input html-element ())  
  #:extra-constructor-name make-input)
```

An `input` is (`make-input` (`listof attribute`))

```
(struct col html-element ())  
  #:extra-constructor-name make-col)
```

A `col` is (`make-col` (`listof attribute`))

```
(struct isindex html-element ())  
  #:extra-constructor-name make-isindex)
```

An `isindex` is (`make-isindex` (`listof attribute`))

```
(struct base html-element ())  
  #:extra-constructor-name make-base)
```

A `base` is (`make-base` (`listof attribute`))

```
(struct meta html-element ())  
  #:extra-constructor-name make-meta)
```

A `meta` is (`make-meta` (`listof attribute`))

```
(struct option html-full ()
  #:extra-constructor-name make-option)
```

An option is (make-option (listof attribute) (listof pcdata))

```
(struct textarea html-full ()
  #:extra-constructor-name make-textarea)
```

A textarea is (make-textarea (listof attribute) (listof pcdata))

```
(struct title html-full ()
  #:extra-constructor-name make-title)
```

A title is (make-title (listof attribute) (listof pcdata))

```
(struct head html-full ()
  #:extra-constructor-name make-head)
```

A head is (make-head (listof attribute) (listof Contents-of-head))

A Contents-of-head is either

- base
- isindex
- alink
- meta
- object
- script
- style
- title

```
(struct tr html-full ()
  #:extra-constructor-name make-tr)
```

A tr is (make-tr (listof attribute) (listof Contents-of-tr))

A Contents-of-tr is either

- `td`
- `th`

```
(struct colgroup html-full ()  
  #:extra-constructor-name make-colgroup)
```

A `colgroup` is `(make-colgroup (listof attribute) (listof col))`

```
(struct thead html-full ()  
  #:extra-constructor-name make-thead)
```

A `thead` is `(make-thead (listof attribute) (listof tr))`

```
(struct tfoot html-full ()  
  #:extra-constructor-name make-tfoot)
```

A `tfoot` is `(make-tfoot (listof attribute) (listof tr))`

```
(struct tbody html-full ()  
  #:extra-constructor-name make-tbody)
```

A `tbody` is `(make-tbody (listof attribute) (listof tr))`

```
(struct tt html-full ()  
  #:extra-constructor-name make-tt)
```

A `tt` is `(make-tt (listof attribute) (listof G5))`

```
(struct i html-full ()  
  #:extra-constructor-name make-i)
```

An `i` is `(make-i (listof attribute) (listof G5))`

```
(struct b html-full ()  
  #:extra-constructor-name make-b)
```

A `b` is `(make-b (listof attribute) (listof G5))`

```
(struct u html-full ()  
  #:extra-constructor-name make-u)
```

An `u` is `(make-u (listof attribute) (listof G5))`

```
(struct s html-full ()  
  #:extra-constructor-name make-s)
```

A `s` is `(make-s (listof attribute) (listof G5))`

```
(struct strike html-full ()  
  #:extra-constructor-name make-strike)
```

A `strike` is `(make-strike (listof attribute) (listof G5))`

```
(struct big html-full ()  
  #:extra-constructor-name make-big)
```

A `big` is `(make-big (listof attribute) (listof G5))`

```
(struct small html-full ()  
  #:extra-constructor-name make-small)
```

A `small` is `(make-small (listof attribute) (listof G5))`

```
(struct em html-full ()  
  #:extra-constructor-name make-em)
```

An `em` is `(make-em (listof attribute) (listof G5))`

```
(struct strong html-full ()  
  #:extra-constructor-name make-strong)
```

A `strong` is `(make-strong (listof attribute) (listof G5))`

```
(struct dfn html-full ()  
  #:extra-constructor-name make-dfn)
```

A `dfn` is `(make-dfn (listof attribute) (listof G5))`

```
(struct code html-full ()  
  #:extra-constructor-name make-code)
```

A `code` is `(make-code (listof attribute) (listof G5))`

```
(struct samp html-full ()
  #:extra-constructor-name make-samp)

A samp is (make-samp (listof attribute) (listof G5))
```

```
(struct kbd html-full ()
  #:extra-constructor-name make-kbd)

A kbd is (make-kbd (listof attribute) (listof G5))
```

```
(struct var html-full ()
  #:extra-constructor-name make-var)

A var is (make-var (listof attribute) (listof G5))
```

```
(struct cite html-full ()
  #:extra-constructor-name make-cite)

A cite is (make-cite (listof attribute) (listof G5))
```

```
(struct abbr html-full ()
  #:extra-constructor-name make-abbr)

An abbr is (make-abbr (listof attribute) (listof G5))
```

```
(struct acronym html-full ()
  #:extra-constructor-name make-acronym)

An acronym is (make-acronym (listof attribute) (listof G5))
```

```
(struct sub html-full ()
  #:extra-constructor-name make-sub)

A sub is (make-sub (listof attribute) (listof G5))
```

```
(struct sup html-full ()
  #:extra-constructor-name make-sup)

A sup is (make-sup (listof attribute) (listof G5))
```

```
(struct span html-full ()
  #:extra-constructor-name make-span)

A span is (make-span (listof attribute) (listof G5))
```

```
(struct bdo html-full ()
  #:extra-constructor-name make-bdo)

A bdo is (make-bdo (listof attribute) (listof G5))
```

```
(struct font html-full ()
  #:extra-constructor-name make-font)

A font is (make-font (listof attribute) (listof G5))
```

```
(struct p html-full ()
  #:extra-constructor-name make-p)

A p is (make-p (listof attribute) (listof G5))
```

```
(struct h1 html-full ()
  #:extra-constructor-name make-h1)

A h1 is (make-h1 (listof attribute) (listof G5))
```

```
(struct h2 html-full ()
  #:extra-constructor-name make-h2)

A h2 is (make-h2 (listof attribute) (listof G5))
```

```
(struct h3 html-full ()
  #:extra-constructor-name make-h3)

A h3 is (make-h3 (listof attribute) (listof G5))
```

```
(struct h4 html-full ()
  #:extra-constructor-name make-h4)

A h4 is (make-h4 (listof attribute) (listof G5))
```

```
(struct h5 html-full ()
  #:extra-constructor-name make-h5)

A h5 is (make-h5 (listof attribute) (listof G5))
```

```
(struct h6 html-full ()
  #:extra-constructor-name make-h6)

A h6 is (make-h6 (listof attribute) (listof G5))
```

```
(struct q html-full ()
  #:extra-constructor-name make-q)

A q is (make-q (listof attribute) (listof G5))
```

```
(struct dt html-full ()
  #:extra-constructor-name make-dt)

A dt is (make-dt (listof attribute) (listof G5))
```

```
(struct legend html-full ()
  #:extra-constructor-name make-legend)

A legend is (make-legend (listof attribute) (listof G5))
```

```
(struct caption html-full ()
  #:extra-constructor-name make-caption)

A caption is (make-caption (listof attribute) (listof G5))
```

```
(struct table html-full ()
  #:extra-constructor-name make-table)

A table is (make-table (listof attribute) (listof Contents-of-table))
```

A Contents-of-table is either

- `caption`
- `col`
- `colgroup`

- `tbody`
- `tfoot`
- `thead`

```
(struct button html-full ())  
  #:extra-constructor-name make-button)
```

A `button` is (`make-button` (listof `attribute`) (listof G4))

```
(struct fieldset html-full ())  
  #:extra-constructor-name make-fieldset)
```

A `fieldset` is (`make-fieldset` (listof `attribute`) (listof Contents-of-fieldset))

A Contents-of-fieldset is either

- `legend`
- G2

```
(struct optgroup html-full ())  
  #:extra-constructor-name make-optgroup)
```

An `optgroup` is (`make-optgroup` (listof `attribute`) (listof `option`))

```
(struct select html-full ())  
  #:extra-constructor-name make-select)
```

A `select` is (`make-select` (listof `attribute`) (listof Contents-of-select))

A Contents-of-select is either

- `optgroup`
- `option`

```
(struct label html-full ())  
  #:extra-constructor-name make-label)
```

A `label` is (`make-label` (listof `attribute`) (listof G6))

```
(struct form html-full ()
  #:extra-constructor-name make-form)

A form is (make-form (listof attribute) (listof G3))
```

```
(struct ol html-full ()
  #:extra-constructor-name make-ol)

An ol is (make-ol (listof attribute) (listof li))
```

```
(struct ul html-full ()
  #:extra-constructor-name make-ul)

An ul is (make-ul (listof attribute) (listof li))
```

```
(struct dir html-full ()
  #:extra-constructor-name make-dir)

A dir is (make-dir (listof attribute) (listof li))
```

```
(struct menu html-full ()
  #:extra-constructor-name make-menu)

A menu is (make-menu (listof attribute) (listof li))
```

```
(struct dl html-full ()
  #:extra-constructor-name make-dl)

A dl is (make-dl (listof attribute) (listof Contents-of-dl))

A Contents-of-dl is either
```

- dd
- dt

```
(struct pre html-full ()
  #:extra-constructor-name make-pre)

A pre is (make-pre (listof attribute) (listof Contents-of-pre))

A Contents-of-pre is either
```

- G9
- G11

```
(struct object html-full ()  
      #:extra-constructor-name make-object)
```

An `object` is `(make-object (listof attribute) (listof Contents-of-object-applet))`

```
(struct applet html-full ()  
      #:extra-constructor-name make-applet)
```

An `applet` is `(make-applet (listof attribute) (listof Contents-of-object-applet))`

A `Contents-of-object-applet` is either

- `param`
- G2

```
(struct map html-full ()  
      #:extra-constructor-name make-map)
```

A `Map` is `(make-map (listof attribute) (listof Contents-of-map))`

A `Contents-of-map` is either

- `area`
- `fieldset`
- `form`
- `isindex`
- G10

```
(struct a html-full ()  
      #:extra-constructor-name make-a)
```

An `a` is `(make-a (listof attribute) (listof Contents-of-a))`

A `Contents-of-a` is either

- `label`
- `G7`

```
(struct address html-full ()  
      #:extra-constructor-name make-address)
```

An `address` is `(make-address (listof attribute) (listof Contents-of-address))`

A `Contents-of-address` is either

- `p`
- `G5`

```
(struct body html-full ()  
      #:extra-constructor-name make-body)
```

A `body` is `(make-body (listof attribute) (listof Contents-of-body))`

A `Contents-of-body` is either

- `del`
- `ins`
- `G2`

A `G12` is either

- `button`
- `iframe`
- `input`
- `select`
- `textarea`

A `G11` is either

- `a`

- `label`
- `G12`

A `G10` is either

- `address`
- `blockquote`
- `center`
- `dir`
- `div`
- `dl`
- `h1`
- `h2`
- `h3`
- `h4`
- `h5`
- `h6`
- `hr`
- `menu`
- `noframes`
- `noscript`
- `ol`
- `p`
- `pre`
- `table`
- `ul`

A `G9` is either

- `abbr`

- acronym
- b
- bdo
- br
- cite
- code
- dfn
- em
- i
- kbd
- map
- pcddata
- q
- s
- samp
- script
- span
- strike
- strong
- tt
- u
- var

A G8 is either

- applet
- basefont
- big
- font

- `img`
- `object`
- `small`
- `sub`
- `sup`
- G9

A G7 is either

- G8
- G12

A G6 is either

- `a`
- G7

A G5 is either

- `label`
- G6

A G4 is either

- G8
- G10

A G3 is either

- `fieldset`
- `isindex`
- G4
- G11

A G_2 is either

- form
- G_3