

# The **anyweb** Hack

Sebastien Mondet

Literate Programming With Anything You May Find

## 1 Compile

It is simple:

```
ocamlc unix.cma anyweb.ml -o anyweb
```

## 2 Running Examples

This document comes from **anyweb**'s source *pip*ed to **bracetax**:

```
anyweb camlbrtxhtml anyweb.ml | \  
  brtx -doc -o index.html -title "The anyweb Source" -link-css anyweb.css
```

if you are curious, here is a version without CSS (i.e. with `-link-css anyweb.css`).

The same way we can make [a PDF](#):

```
anyweb camlbrtxlatex anyweb.ml | \  
  brtx -latex -doc -o anyweb.tex -title "The anyweb Source"  
pdflatex anyweb
```

This other example *documents* a **Coq .v** file:

```
anyweb coqbrtxhtml subset_notes.v | \  
  brtx -o coq_example.html -doc -link-css anyweb.css  
anyweb coqbrtxlatex subset_notes.v | \  
  brtx -latex -o coq_example.tex -doc -use-package coqdoc  
pdflatex coq_example
```

The results are some pieces of **HTML** and **PDF**. (these are some notes taken while *doing CPDT* ... which is worth reading!).

## 3 The Code

We do not print the code for:

```
val split : string -> string -> string * string  
val line_opt : in_channel -> string option  
val feed : cmd:string -> input:string -> string
```

which can be found in **ExtLib** and **Camlmix**'s toolbox.

### 3.1 The environment

```
type environment = {
  start_token : string;
  on_begin : out_channel -> unit;
  on_text : out_channel -> string -> unit;
  on_change : out_channel -> unit;
  end_token : string;
  on_end : out_channel -> unit;
  contains : string list;
}
let environment
  ?(on_begin = fun o -> ())
  ?(on_text = output_string)
  ?(on_change = fun o -> ())
  ?(on_end = fun o -> ())
  start_token end_token contains =
{ on_begin; on_text; on_change; on_end;
  start_token; end_token; contains }
```

### 3.2 The transformation

```
let split_first s l current =
  List.fold_left
    (fun m k ->
      match m, k with
      | None, x -> x
      | x, None -> x
      | Some (_, _, mb, _), Some (_, _, kb, _) ->
        if String.length mb <= String.length kb then m else k)
    None
    ((match split_opt s current.end_token with
      None -> None | Some (b, a) -> Some (true, current, b, a)) ::
      (List.map (fun env ->
        match split_opt s env.start_token with
        | None -> None
        | Some (b, a) -> Some (false, env, b, a)) l))

let transform_environments in_chan out_chan =
  let rec loop stack current_text =
    match stack with
    | env :: l ->
      let inside = List.map (fun x -> List.assoc x environments) env.contains
```

```

in
  begin match split_first current_text inside env with
  | Some (true, s, before, after) -> (* unstack *)
    env.on_text out_chan before;
    env.on_end out_chan;
    loop 1 after
  | Some (false, s, before, after) -> (* stack *)
    env.on_text out_chan before;
    env.on_change out_chan;
    s.on_begin out_chan;
    loop (s :: stack) after
  | None ->
    env.on_text out_chan current_text;
    begin match line_opt in_chan with
    | Some line ->
      loop stack line
    | None -> env.on_end out_chan; ()
    end
  end
  | [] ->
    failwith
      (sprintf "Unstacked too much, do not know what to do now: %S"
        current_text)
in
let toplevel = (snd (List.hd environments)) in
  toplevel.on_begin out_chan;
  loop [ toplevel ] "";
  ()

```

### 3.3 Available environments

First, a *complicated* one, used for testing:

```

let test_environments = [
  "brackets",
  environment
    ~on_begin:(fun o -> output_string o "(START_BRACKETS)")
    ~on_text:(fun o s -> output_string o (String.uppercase s))
    ~on_end:(fun o -> output_string o "(END_BRACKETS)")
    "[[" "]" [ "braces" ];
  "braces",
  environment
    ~on_begin:(fun o -> output_string o "(START_BRACES)")
    ~on_text:(fun o s -> output_string o (String.uppercase s))
    ~on_end:(fun o -> output_string o "(END_BRACES)")
    "{{" "}" [ "LTGTs"; "parens" ];
  "LTGTs",
  environment

```

```

~on_begin:(fun o -> output_string o "(START_LTGTs)")
~on_text:(fun o s -> output_string o (String.uppercase s))
~on_end:(fun o -> output_string o "(END_LTGTs)")
"<<" ">>" [];
"parens",
environment
~on_begin:(fun o -> output_string o "(START_PARENS)")
~on_text:(fun o s -> output_string o (String.uppercase s))
~on_end:(fun o -> output_string o "(END_PARENS)")
"((( " " )))" [ "brackets" ];
]

```

A function to create two functions: one which stores in a buffer, and another one which gives the contents of the buffer to the argument and clears the *internal* buffer.

```

let bufferise_and_do f =
  let buffer = Buffer.create 42 in
  ((fun o s -> Buffer.add_string buffer s),
   (fun o ->
    let stuff_done = f (Buffer.contents buffer) in
    output_string o stuff_done;
    Buffer.clear buffer))

```

This function name is self-explanatory:

```

let is_whitespace s =
  try
    String.iter (function
      | ' ' | '\n' | '\r' | '\t' -> ()
      | c -> raise Exit) s;
    true
  with Exit -> false

```

The few tricks needed now here are:

- The `coqdoc` command line: we use `cat` to dump `stdin` to a file, and then we call `coqdoc`.
- We have to write things like `"(*" ^ "B" or "B" ^ "*)"` to allow `anyweb` to run on its own source.

This gives the `coqbrtx` transformer:

```

let coqbrtx fmt =
  let coqdoc =
    sprintf
      "cat > /tmp/ttt.v ; coqdoc -s --parse-comments --stdout \
      --body-only --no-index %s /tmp/ttt.v"
    (match fmt with 'html -> "--html" | 'latex -> "--latex") in
  [

```

```

"coq",
(let on_text, on_end =
  bufferise_and_do (fun input ->
    if is_whitespace input then "# Removed whitespace\n"
    else
      "{bypass endanywebbypass}" ^ (feed ~cmd:coqdoc ~input)
      ^ "{endanywebbypass}") in
  environment ~on_text ~on_end ~on_change:on_end
  "[coq[" "]"coq]" [ "bracetax" ]]);
"bracetax", environment ("(" ^ "B") ("B" ^ ")") [ "coq" ];
]

```

And similarly the `camlbrtx` one:

```

let camlbrtx fmt = [
  "caml",
  (let on_text, on_end =
    let cmd =
      sprintf "source-highlight -s caml -f %s"
      (match fmt with 'html -> "xhtml" | 'latex -> "latex") in
    bufferise_and_do (fun input ->
      if is_whitespace input then "# Removed whitespace\n"
      else
        "{bypass endanywebcode}" ^ (feed ~cmd ~input) ^ "{endany" ^
"webcode}") in
    environment ~on_text ~on_end ~on_change:on_end
    ("[" ^ "ca" ^ "ml[" "]"ca" ^ "ml]" [ "bracetax" ]]);
  "bracetax", environment ("(" ^ "B") ("B" ^ ")") [ "caml" ];
]

```

### 3.4 The “main” function

```

let () =
  let lang =
    try match Sys.argv.(1) with
    | "coqbrtxhtml" -> coqbrtx 'html
    | "coqbrtxlatex" -> coqbrtx 'latex
    | "camlbrtxhtml" -> camlbrtx 'html
    | "camlbrtxlatex" -> camlbrtx 'latex
    | _ -> test_environments
    with e -> test_environments in
  let i = try open_in Sys.argv.(2) with e -> stdin in
  let o = try open_out Sys.argv.(3) with e -> stdout in
  transform lang i o;
  close_in i; close_out o

```

## 4 To-Do List

- command-line-forged transformers